Universidade de São Paulo

Instituto de Física

Implementação de um protocolo *Dynamic Clamp* em sistema Linux em tempo real para a produção de condutâncias artificiais em neurônios biológicos e eletrônicos

Rogério Mazur

Orientador: Prof. Dr. Reynaldo Daniel Pinto

Dissertação apresentada ao Instituto de Física da Universidade de São Paulo para a obtenção do título de Mestre em Ciências.

São Paulo
2006

FICHA CATALOGRÁFICA
Preparada pelo Serviço de Biblioteca e Informação
do Instituto de Física da Universidade de São Paulo

Mazur, Rogério
Implementação de um protocolo Dynamic Clamp em sistema Linux em tempo real para a produção de condutâncias artificiais em neurônios biológicos e eletrônicos. São Paulo, 2006.

Dissertação (Mestrado) - Universidade de São Paulo. Instituto de Física, Depto. Física Geral.
Orientador: Prof. Reynaldo Daniel Pinto

Área de Concentração: Física
.
Unitermos: 1.Plasticidade neuronal; 2. Potenciais de ação; 3. Instrumentação (Física); 4. Física computacional 5. Linux

USP/IF/SBI-077/2006

Agradecimentos

Tenho uma profunda gratidão ao meu orientador, Dr. Reynaldo D. Pinto, por sua orientação e apoio, pela sua compreensão nos meus momentos dificeis e por ter sido grande incentivador.

Ao Prof. Dr. José Carlos Sartorelli do LFNL pelo seu apoio.

Ao Dr. Thomas Nowotny do Institute for Nonlinear Science, UCSD, pela ajuda prestada.

À minha familia, pelos apoios e incentivos.

À Dayenne por estar junto em todos os momentos felizes e dificeis sempre incentivando, pelo exemplo de pessoa que ela sempre será, sem ela esse trabalho não seria possível, dedico esse trabalho a ela, meu eterno amor, sempre estará comigo.

À todos os colegas do LFNL que proporcionaram um ambiente cordial de apoio e incentivo.

À todos os amigos que que contribuiram para a realização desse trabalho, e para torná-lo mais agradavel.

Aos funcionários do Ifusp pelo apoio e compania que tiveram.

Ao grande amigo Hernán Joel Cervantes Rodríguez's pela contribuicão no trabalho e amizade.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e à Fundação de Amparo à Pesquisa do estado de São Paulo (FAPESP), pelo apoio financeiro.

Este trabalho é dedicado, com muito amor, in memoriam de Dayenne de F. Lançoni Gama que sempre incentivou na vida e realização desse projeto.

"O que fazemos em vida ecoa na eternidade"

RESUMO

O protocolo conhecido como Dynamic Clamp consiste em utilizar um computador para introduzir condutâncias artificiais em um neurônio biológico. O modo como estas condutâncias dependem da voltagem da membrana ou do tempo são modelado por equações diferenciais que são integradas em tempo real por um computador conectado ao neurônio biológico. Resumidamente, o computador tem acesso ao potencial de membrana dos neurônios através de eletrodos intracelulares conectados a conversores analógico-digitais (ADCs), calcula as correntes a serem injetadas nos neurônios e produz os sinais de saída através de conversores digital-analógicos (DACs) que produzem a injeção das correntes nos eletrodos intracelulares. De um certo modo, o Dynamic Clamp utiliza os neurônios como simuladores, permitindo investigar a importância de um tipo de condutância para a atividade elétrica de um neurônio, assim como determinar o efeito produzido pelas sinapses em uma rede, combinando o controle e flexibilidade de uma simulação no computador com a acurácia e o realismo de um experimento em eletrofisiologia.

Descrevemos a implementação de um protocolo de Dynamic Clamp utilizando um computador pessoal tipo IBM-PC que permitiu contornar 3 das principais limitações que apresentam alguns dos programas de Dynamic Clamp comerciais/gratuitos disponíveis atualmente:

(a) Garantia de que o sistema roda em tempo real – nossa implementação é baseada em um programa de Dynamic Clamp que roda em uma plataforma Linux Real-Time que além de controlar os experimentos em tempo real consiste em software livre com codigo fonte aberto e que pode ser instalado gratuitamente;

(b) Não necessita de hardware de aquisição de dados dedicado para eletrofisiologia – utilizamos uma placa ADC/DAC comercial comum marca National Instruments modelo PCI-MIO16E4. Com o driver COMEDI instalado para placas de aquisição de dados Linux, a maioria das placas ADC/DAC tipo PCI disponíveis no mercado podem ser utilizadas em implementações futuras;

(c) Aumentar o número de neurônios que podem ser conectados simultaneamente - desenvolvemos um circuito demultiplex analógico que permite controlar até 8 neurônios biológicos/artificiais a partir das duas saídas analógicas que as placas DAC comerciais possuem e ainda atingir frequências de atualização da corrente de até 3 kHz (para 8 correntes de saída).

Apresentamos os resultados de diversos testes que fizemos usando o programa adaptado e o circuito demultiplex para produzir sinapses em tempo real e conectar diversos neurônios artificiais em pequenas redes. Também mostramos alguns resultados preliminares obtidos com a primeira implementação de um modelo de neurônio estocástico tipo Hodgkin-Huxley em um programa de Dynamic Clamp.

ABSTRACT

The Dynamic Clamp protocol consists in using a computer to introduce artificial conductances in a biological neuron. The voltage- and time-dependency of each conductance is modeled by differential equations integrated in real-time by the computer connected to the biological neurons. In short, the computer executes a 3-phase cycle in which the membrane potential of the neurons is measured by intracellular electrodes and digitized by analog-to-digital converters (ADCs), the currents are calculated based in the digitized membrane potentials and current signals are generated by digital-to-analog converters (DACs). These currents are actually injected in the neurons by other intracellular electrodes. In some extent the Dynamic Clamp uses the neurons as simulators, allowing one to investigate the role of a specific conductance in the intrinsic activity of a neuron as well as to look for the effects of a synapse in the behavior of a small network. The Dynamic Clamp combines the control and flexibility of a computer simulation with the reality of an experiment in electrophysiology.

We describe an implementation of a Dynamic Clamp protocol that allowed us to surmount 3 of the main drawbacks present in some commercial/freely available Dynamic Clamp programs:

(a) Runs in real time – our implementation is based in a program that runs in a Real-Time Linux platform. This operating system not only ensures the experiments will be controlled in real time but also consists in open source software that can be freely downloaded and installed;

(b) No need of special electrophysiology acquisition hardware – we used a commercial ADC/DAC acquisition board model PCI-MIO16E4 from National Instruments. With the COMEDI Linux package driver that is used most of the PCI commercial ADC/DAC boards can be used in future implementations with no change needed in the program itself.

(c) We can connect more than two neurons with artificial synapses – we developed an analog demultiplex circuit that allowed us to control simultaneously up to 8 biological/artificial neurons from the two analog outputs available in most of the commercial ADC/DAC boards and we could still reach current update rates of about 3 kHz (for 8 current outputs enabled).

We present the results of several tests we performed using the program adapted to control the analog demultiplex to establish synapses and to connect several artificial neurons in small neural networks. Preliminary results from the first implementation of a stochastic whole cell Hodgkin-Huxley model neuron in a real time Dynamic Clamp program are also shown.

Índice das figuras

Índice

1. Introdução

O sistema nervoso dos animais evoluiu de modo a resolver problemas muito complexos, relacionados com o controle do comportamento em ambientes naturais multidimensionais, não-estacionários e imprevisíveis. O próprio sistema nervoso é um dos sistemas mais complexos existentes na natureza. Esta complexidade é o resultado não apenas da interação de um grande número de elementos não-lineares e da sofisticada arquitetura das conexões, mas também da flexibilidade e plasticidade desses elementos (Kandel at al., 1991; Levitan & Kaczmarek, 1997). Todas as informações sensoriais que o sistema nervoso processa e todos os comportamentos motores gerados pelos animais são codificados na forma de sinais elétricos produzidos e transmitidos pelos neurônios (Rieke et al.,1997).

A atividade elétrica neural é produzida por uma grande população de diferentes canais iônicos que os neurônios possuem em sua membrana celular (Hille, 2001). Estes canais podem permitir ou não a passagem seletiva de íons, dependendo do potencial elétrico em que a célula se encontra, da presença de substâncias químicas ou de alguma ação mecânica externa. Os canais iônicos conferem não apenas a característica especial de excitabilidade da célula que permite produzir os impulsos chamados de potenciais de ação, mas também permitem propagar estes sinais para outros neurônios através de regiões das membranas conhecidas como sinapses.

Mesmo em redes neurais muito simples, que possuem pequeno número de neurônios com detalhes fisiológicos bastante conhecidos, não se pode dizer que se compreenda, de forma rigorosa, como um determinado comportamento é produzido pela rede. Isto é, em um nível reducionista, as propriedades fisiológicas das células e das sinapses foram estabelecidas, mas descobriu-se que a operação total da rede neural é uma "propriedade emergente" do sistema, que é mais que a simples soma de suas partes (Selverston et al., 2000). Um dos exemplos mais populares de tais redes simples que se comportam como sistemas complexos são os centros geradores de padrões (CPGs) do sistema nervoso estomatogástrico de crustáceos (Mulloney & Selverston, 1974; Selverston & Moulins, 1986).

Os CPGs mais simples, presentes tanto em invertebrados como em vertebrados, são pequenas redes neurais (algumas dezenas de células em invertebrados) que se desenvolveram para produzir padrões motores espaço-temporais rítmicos sem a necessidade de estímulos sensoriais. Estes padrões controlam a contração de músculos

usados para executar atividades repetitivas como, por exemplo, andar, correr, nadar, mastigar. Evidentemente, os padrões gerados são fortemente influenciados não apenas por estímulos sensoriais, mas também por substâncias neuromoduladoras, porém as características fundamentais dos padrões são determinadas pelas configurações das conexões sinápticas e pelas propriedades biofísicas dos neurônios individuais.

A representação matemática do comportamento elétrico de neurônios usando modelos que, baseados em medidas experimentais eletrofisiológicas (Hodgkin & Huxley, 1952; Hindmarsh & Rose, 1984), permitem manipular a dinâmica de condutâncias específicas e testar diferentes hipóteses sobre o comportamento do neurônio isolado, constitui uma poderosa ferramenta para estudar muitas questões de interesse em neurociência. Entre elas podemos ressaltar: como as condutâncias individuais moldam o comportamento de uma célula ou qual é a importância de cada uma das propriedades individuais dos neurônios na determinação do comportamento global de uma rede neural complexa (Marder, 1998). Mesmo nas menores e mais simples redes neurais biológicas estas questões estão longe de serem respondidas por que os próprios neurônios são sistemas altamente complexos e não-lineares (Holden, 1997; Rinzel & Ermentrout, 1998; Izhikevich, 2000).

Modelos matemáticos realistas da atividade dos neurônios, além de constituir uma maneira de testar diversas hipóteses (muitas vezes impossíveis de testar em um neurônio vivo) sobre os processos que colaboram para produzir um comportamento experimental particular, podem ser usados para fazer previsões e orientar experimentos que revelem novas propriedades das células (Marder, 1998; Dayan & Abbott, 2001).

Um dos métodos mais tradicionais usados para se alterar a atividade elétrica de um neurônio ou perturbar o padrão de atividade de uma rede neural biológica é injetar uma corrente contínua no neurônio usando "*Current Clamp*", um método que tanto permite inibir como excitar um neurônio. As principais ferramentas usadas para investigar as características de uma particular condutância em um neurônio são os métodos conhecidos como "*Voltage Clamp*" e "*Current Clamp*" que consistem em injetar uma corrente variável no neurônio (ou numa pequena área retirada de sua membrana) para manter fixa a voltagem (Kandel et al, 1991; Levitan & Kaczmareck, 1997). Embora essenciais para a compreensão das propriedades elétricas e temporais de uma determinada condutância, estes métodos têm pouca utilidade para estudar o comportamento coletivo de todas as condutâncias que determinam como um neurônio se comporta individualmente ou em um circuito.

A maioria dos experimentos que utilizam *voltage* ou *patch-clamp* fixa ou limita os

valores de voltagem que a membrana do neurônio em estudo pode apresentar e, normalmente, agentes farmacológicos são utilizados para isolar um único tipo de condutância. Desse modo, estes métodos clássicos não permitem, durante a operação normal do neurônio, estudar as alterações das condutâncias da membrana da célula devido às suas propriedades elétricas intrínsecas ou em resposta aos sinais recebidos pelas sinápses com outros neurônios.

A modelagem matemática do comportamento de um neurônio através de um sistema de equações diferenciais (sistema dinâmico) não é útil apenas para simular o comportamento elétrico intrínseco de uma célula nervosa, mas também se aplica às sinapses (Desthexhe et al., 1994) , que podem ser modeladas como condutâncias que dependem do comportamento de dois neurônios (um neurônio pré-sináptico e um neurônio pós-sináptico).

O método conhecido como Dynamic Clamp consiste em utilizar um computador para introduzir condutâncias artificiais em um neurônio biológico (Sharp et al., 1992, 1993, 1996; Pinto et al., 2001; Butera et al., 2001; Dorval et al., 2001; Prinz et al., 2004). O modo como estas condutâncias dependem da voltagem da membrana ou do tempo é especificado por equações matemáticas que são integradas em tempo real por um computador conectado ao neurônio biológico. De um certo modo, o Dynamic Clamp utiliza os neurônios como simuladores, permitindo investigar a importância de um tipo de condutância para a atividade elétrica de um neurônio, assim como determinar o efeito produzido pelas sinapses em uma rede, combinando o controle e flexibilidade de uma simulação no computador com a acurácia e o realismo de um experimento em eletrofisiologia. Uma das mais promissoras aplicações do Dynamic Clamp tem sido conectar modelos matemáticos simulados em tempo real com tecido vivo através de sinapses artificiais, promovendo a troca de informação entre os neurônios biológicos e um computador digital que é entendido pelos neurônios como se fizesse parte do circuito biológico.

Esta troca de informações entre experimentos com neurônios vivos e modelos matemáticos pode abrir muitas novas possibilidades. Entre outras aplicações, modelos analógicos ou modelos de neurônios virtuais, integrados em tempo real em computadores digitais, podem interagir com tecido vivo e produzir redes neurais híbridas (Szücs *et al*., 2000; Pinto *et al*., 2000; Prinz, 2004) que podem auxiliar a compreender melhor como as redes biológicas funcionam (Selverston *et al*., 2000), restaurar circuitos danificados (Szücs *et al*., 2000), construir circuitos com novas propriedades (Ayers, 2004), ou até alimentar uma rede neural viva com dados sensoriais vindos de dispositivos artificiais

comandados pela rede biológica como acontece nas interfaces cérebro-máquina (Nicolelis, 2003).

É neste efervescente contexto multidisciplinar que apresentamos a presente dissertação, onde relatamos a implementação de um protocolo Dynamic Clamp em ambiente Real Time Linux (http://www.bu.edu/ndl/rtldc.html) baseado no trabalho original de Dorval, Christini & White (Dorval et al., 2001) e o desenvolvimento de um circuito eletrônico demultiplexador analógico capaz de produzir sinais de corrente para conectar até 8 neurônios artificiais e/ou biológicos.

O texto desta dissertação encontra-se organizado de modo a apresentar inicialmente, de modo simplificado, o funcionamento elétrico de um neurônio biológico (Capítulo 2) e os modelos matemáticos que foram usados para implementar diversos tipos de neurônios artificiais (Capítulos 3 a 5). Assim, um leitor já familiarizado com estes assuntos pode ir diretamente aos Capítulos 6 a 8, onde introduzimos o protocolo Dynamic Clamp e os modelos usados nas sinapses artificiais, descrevemos nossa implementação do protocolo e os testes realizados.

2. O Neurônio Biológico e a Equação da Membrana

Os neurônios podem apresentar uma grande variedade de formas e tamanhos, mas possuem determinados aspectos em comum. A célula nervosa é formada por um corpo celular que contém o núcleo e um grande número de fibras finas que se prolongam a partir desse corpo, conforme mostrado na Figura 1. Cada neurônio geralmente tem uma única fibra longa (o axônio), que em grandes animais pode ter vários metros de comprimento, e um grande número de fibras mais curtas (dendritos), que são intensamente ramificados e na maioria das vezes possuem menos de 1mm de comprimento (Kandel et al., 1991; Levitan & Kaczmareck, 1997; Schmidt-Nielsen, 1996).

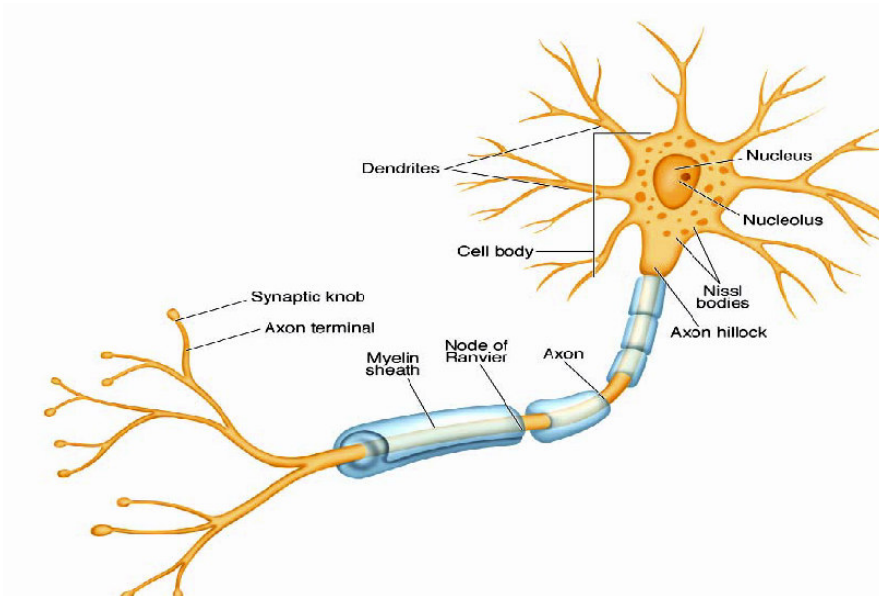


Figura 1: Esquema de um neurônio multipolar. No corpo celular (ou soma), de onde partem os dendritos e o axônio, ocorrem as principais reações relativas ao metabolismo do neurônio, tais como a respiração celular e a síntese das principais substâncias vitais. O axônio é o principal caminho por onde o sinal do neurônio é transmitido aos neurônios conectados a ele. O potencial de ação é gerado na região de interface entre o axônio e o soma, chamada cone de implantação ("axon hilock"). Os dendritos são responsáveis principalmente pela transmissão de sinais provenientes de outros neurônios até o corpo celular.

Nos vertebrados, o corpo celular do neurônio tem, geralmente, menos de 0,1 mm de diâmetro e, as fibras, menos de 0,01 mm de espessura. Em invertebrados, tanto o corpo celular como o diâmetro axonal podem atingir até 1 mm em casos extremos.

As fibras longas, os axônios, são as linhas de comunicação entre neurônios ou entre um neurônio e o tecido muscular. O que comumente é conhecido como um nervo, ou tronco nervoso, consiste de centenas ou milhares de axônios, cada um com origem em

um neurônio distinto. Um nervo não possui corpos celulares; estes são encontrados no sistema nervoso central ou em grupamentos especiais conhecidos como gânglios.

Os pontos nos quais as células e suas extensões fazem contato com outras células nervosas são denominados sinapses e uma única célula nervosa pode, através das sinapses, estar ligada a milhares de outros neurônios. Em sinapses químicas (as mais comuns) a transmissão de um impulso envolve a liberação de um neurotrasmissor (sinal químico) e pode ocorrer apenas em uma direção, do axônio para a célula adjacente, e não na direção contrária.

A membrana celular dos neurônios (e de qualquer outra célula biológica) é principalmente composta por uma bicamada de fosfolipídios (gorduras) que isola (cargas elétricas) o conteúdo intracelular do meio extracelular. Esta membrana tem tipicamente uma espessura de 3 a 6 nm e possui também diversas proteínas imersas que a atravessam, formando um mosaico de lipídios e proteínas, conforme mostrado na Figura 2.

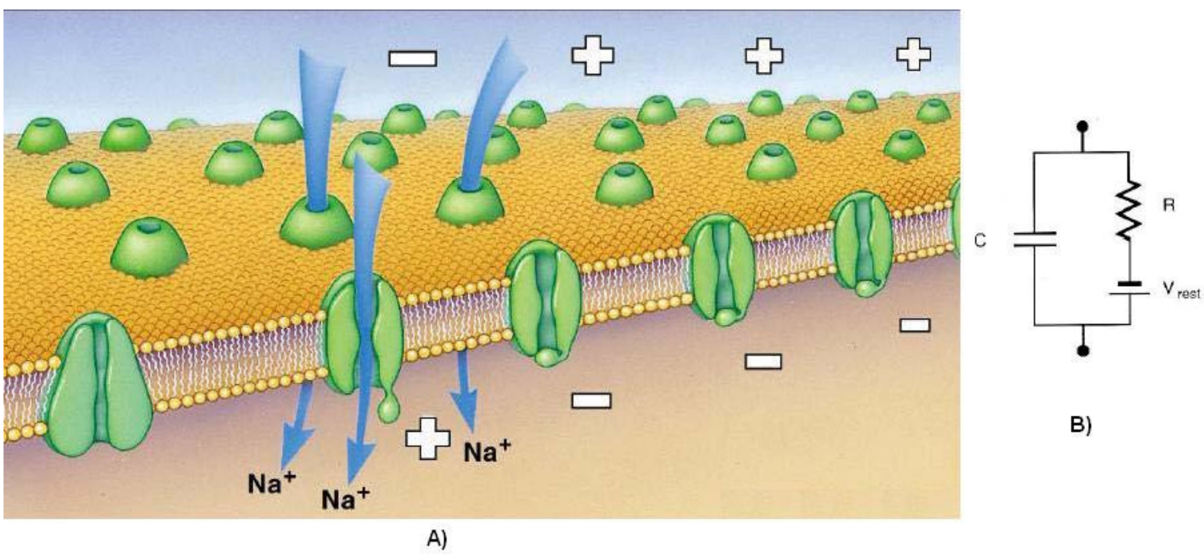


Figura 2: A) Representação de uma secção da membrana celular onde estão imersos os canais iônicos, poros que permitem a passagem seletiva de íons através da membrana dependendo do valor do potencial de membrana. B) Circuito que modela a atividade elétrica da membrana celular: a membrana funciona como um capacitor de placas paralelas conectado a uma bateria e um resistor. A bateria representa o potencial elétrico devido à diferença de concentração iônica nos meios intra e extracelular e o resistor representa o canal iônico e a resistência que ele proporciona à passagem de íons(Kandel et al., 1991).

Quando penetramos em uma célula qualquer (inclusive células de vegetais) com um “fio” ligado a um voltímetro, logo percebemos que existe uma diferença de potencial entre o meio externo e o meio intracelular, chamada de potencial de membrana. A

diferença entre as “células normais” e os neurônios (células nervosas), é que estes podem alterar ativamente seu potencial de membrana e usar estas alterações para transmitir informação. Quando nenhum sinal está sendo gerado, o potencial de membrana atinge um equilíbrio dinâmico, permanecendo negativo no meio intracelular em relação ao meio extracelular. Este potencial de equilíbrio é chamado de potencial de repouso.

Se considerarmos uma diferença de potencial de 60 mV (potencial de repouso típico em neurônios) entre as superfícies de uma membrana celular de 6 nm de espessura, temos um campo elétrico V/d ~ 10 kV/mm, ou seja: um campo elétrico uma ordem de grandeza maior que o campo necessário para produzir uma descarga elétrica (faísca) no ar seco (1 kV/mm). Portanto, a membrana celular é um excelente isolante.

O potencial de repouso é produzido por uma distribuição desigual de íons dentro e fora das células: dos tipos de íons mais abundantes encontrados em ambos os lados da membrana, sódio (Na^+), cloro (Cl^-) e cálcio (Ca^{+2}) são encontrados em maior concentração no meio extracelular, enquanto que potássio (K^+) e ânions orgânicos (A^-) são encontrados em maior concentração no meio intracelular.

Muitas das proteínas que atravessam a membrana constituem poros que permitem a passagem passiva de íons por seu interior. Estas proteínas são chamadas canais iônicos e podem possuir mais duas características importantes:

(i) seletividade a um determinado tipo de íon;

(ii) através de mudanças conformacionais, podem abrir ou fechar em resposta a sinais elétricos, mecânicos ou químicos.

A membrana celular dos neurônios é normalmente permeável apenas a íons K^+, pois possui um grande número de canais iônicos seletivos a potássio que estão sempre abertos. Há também canais seletivos a Na^+ e a Cl^- mas consideremos inicialmente que estes estejam todos fechados e que a soma total de cargas positivas e negativas é igual dentro e fora do neurônio e, portanto seu potencial de membrana é nulo. Como os canais de potássio estão abertos, íons K^+ irão fluir para fora da célula na direção de menor gradiente de concentração. Ao saírem, os íons K^+ produzem um desequilíbrio de cargas elétricas (maior número de cargas positivas fora que dentro do neurônio), gerando um campo elétrico que se opõe ao movimento de mais cargas positivas para fora da célula. O valor do potencial elétrico necessário para opor completamente o fluxo de íons X devido à diferença de concentração é chamado de potencial de Nernst (Bear et al.,2002) do íon X e é dado por:

$$E_x = \frac{RT}{zF} \ln \frac{[X^+]_{ext}}{[X^+]_{\text{int}}}$$

Equação de Nert (Bear et al., 2002)

onde R é a constante dos gases, T a temperatura em Kelvin, z é a carga do íon e F é o Faraday (quantidade de carga de um mol de íons monovalentes). Entretanto, o número total de íons K^+ necessário para estabelecer o potencial é muito pequeno quando comparado ao número de íons nos meio intracelular e extracelular, ou seja, o potencial de membrana é estabelecido sem provocar mudanças significantes no gradiente de concentração.

Se uma membrana for permeável a apenas um tipo de íon, o potencial de repouso desta membrana será igual ao potencial de Nernst do íon. Por exemplo, consideremos o caso do axônio gigante da lula, uma das primeiras membranas nervosas estudadas, que possui boa permeabilidade à passagem de íons potássio e $[K^+]_{int} = 20[K^+]_{ext}$. O potencial de Nernst do potássio (E_k) para esta membrana é -75 mV. Introduzindo um eletrodo e medindo o potencial real, um valor de -70mV é encontrado, bastante próximo do valor de E_k.

Como além dos íons K^+ a membrana é permeável a outros íons, como Na^+ e Cl^- (porém bem menos permeável que ao K^+) o potencial de repouso da membrana não é exatamente igual ao E_k. O potencial de repouso Vm pode ser obtido a partir das permeabilidades (p_i) da membrana e das concentrações de cada um dos íons usando a expressão de Goldman:

$$V_m = \frac{RT}{zF} \ln \frac{[K^+]_{ext} + \left(\frac{p_{Na}}{p_K}\right)\left[Na^+\right]_{ext} + \left(\frac{p_{Cl}}{p_K}\right)\left[Cl^-\right]_{\text{int}}}{[K^+]_{\text{int}} + \left(\frac{p_{Na}}{p_K}\right)\left[Na^+\right]_{\text{int}} + \left(\frac{p_{Cl}}{p_K}\right)\left[Cl^-\right]_{ext}}$$

Expressão de Goldman (Kandel et al., 1991)

Como Vm não é igual ao potencial de Nernst de nenhum dos íons individuais, a membrana permanece em um estado de equilíbrio dinâmico, em que os íons permanecem se movendo em sentido contrário a seu gradiente de concentração, porém Vm permanece constante, pois a corrente total (devida a soma das correntes iônicas de cada tipo de íon) é nula.

Como há corrente iônica mesmo durante o “repouso” é necessário algum

mecanismo para manter as diferenças de concentração dos íons, evitando que elas se anulem. Esta tarefa é executada nos neurônios biológicos por um tipo especial de proteínas que existe na membrana, chamadas transportadores iônicos ou bombas iônicas. A mais comum destas proteínas é a sódio-potássio ATPase ou bomba de sódio-potássio. Esta proteína usa um ATP (forma celular de energia) para em um ciclo (que envolve algum tipo de movimento ou rotação da proteína) bombear dois íons Na^+ para fora do neurônio, ao mesmo tempo em que três íons K^+ são bombeados para dentro (Levitan & Kaczmareck, 1997), conforme mostrado na Figura 3.

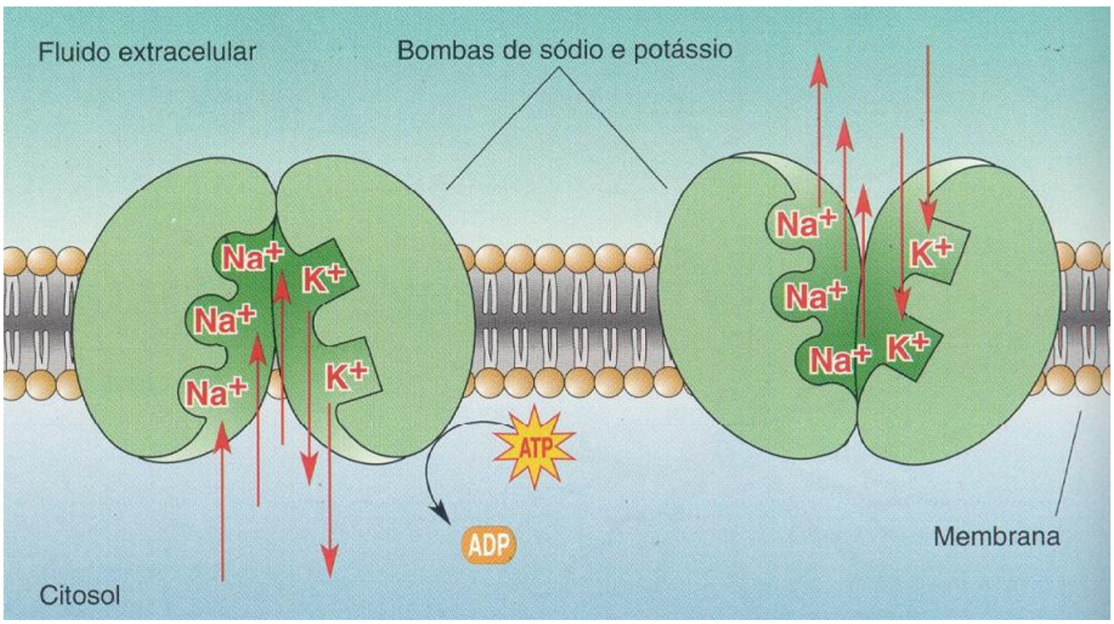


Figura 3: Bomba de sódio-potássio: a energia da hidrólise do ATP é usada para mover três íons de sódio de dentro para fora, e simultaneamente, dois íons de potássio de fora para dentro da celula. Este mecanismo se encarrega de manter as diferenças de concentrações dos íons, fundamentais na determinação do potencial de membrana celular. (Kandel et al., 1991)

Como o esqueleto da membrana dos neurônios é formado por uma bicamada isolante de 3 a 6 nm de espessura e há uma diferença de potencial, os íons que produzem o potencial de membrana tendem a se concentrar nas superfícies da membrana. Sempre que um isolante fino separa cargas elétricas, ele atua como uma capacitância. Ou seja, podemos associar uma capacitância C à membrana celular. Nesse caso, a carga total acumulada nas superfícies é dada por:

$$Q = C.Vm$$

Para mudar a voltagem da membrana uma corrente elétrica deve ser aplicada. Essa corrente capacitiva é obtida diferenciando de ambos os lados a equação da carga total:

$$\frac{dQ}{dt} = C * \frac{dVm(t)}{dt}$$

Mas como dQ/dt = I_{total}, temos para o potencial de membrana:

$$\frac{dVm(t)}{dt} = \frac{1}{C}\sum Ion$$

Como o corpo celular, axônio e dendritos são estruturas extensas, nem a corrente iônica (que depende no número de canais iônicos locais), nem o potencial de membrana são os mesmos ao longo de toda a membrana. Ou seja, o potencial é uma grandeza que assume diferentes valores locais em cada ponto da membrana: $V_m = V_m(\mathbf{r}\ ;\ t)$, onde **r** é o vetor que indica uma posição na membrana.

3. Modelo de Hodgkin-Huxley Determinístico

Os neurônios presentes nos mais diversos animais, têm outra característica em comum além do potencial de repouso: são capazes de produzir impulsos rápidos e estereotipados, como os mostrados na Figura 4, conhecidos como potenciais de ação, que se propagam pelo axônio. O primeiro modelo capaz de explicar como um neurônio produz um impulso nervoso foi desenvolvido por Hodgkin e Huxley no meio do século passado (Hodgkin & Huxley, 1952).

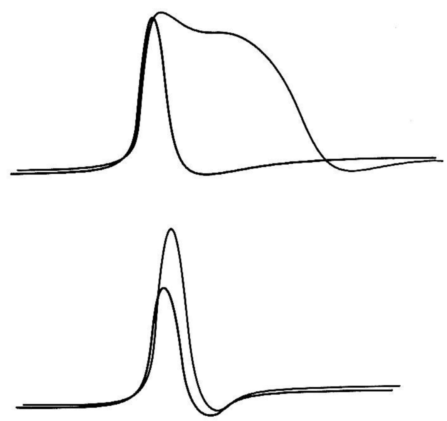


Figura 4: Formato dos potenciais de ação de diferentes neurônios. Quando a atividade elétrica de diferentes neurônios é medida, potenciais de ação de diferentes amplitudes e durações são observados. (Kandel et al., 1991)

Os experimentos que levaram ao modelo de Hodgkin-Huxley foram feitos com o axônio gigante da lula *Loligo*. Este axônio tem dimensões gigantescas (para uma fibra nervosa), com aproximadamente meio milímetro de diâmetro (um axônio típico do córtex de mamíferos tem diâmetro 3 ordens de grandeza menor). Para eliminar a complexidade introduzida pela distribuição espacial dos canais iônicos, um fio axial condutor foi introduzido no axônio e utilizado para manter fixo o potencial elétrico através da membrana axonal (Figura 5). Essa técnica ficou conhecida como "space clamp" pois elimina a dependência da posição onde é medido o potencial da membrana. Deste modo, o axônio se comporta como um capacitor, isopotencial em toda sua extensão. Hodgkin e Huxley (HH) também usaram uma técnica que ficou conhecida como "voltage clamp", desenvolvida por Kenneth Cole em 1940, em que a membrana do axônio é submetida a um potencial elétrico fixo (V_{clamp}) e a corrente elétrica (I_m) necessária para manter este potencial é medida. Desta forma, variando V_{clamp} é possível construir uma curva I_m vs. V_{clamp} como é normalmente feito para descobrir as características de componentes elétricos.

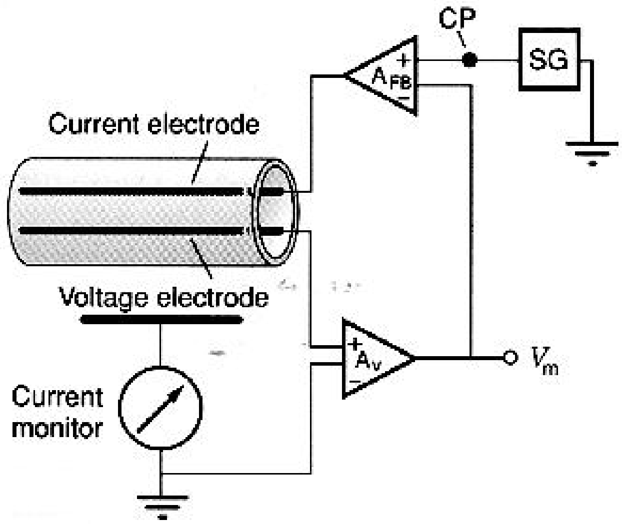


Figura 5: Esquema do aparato utilizado para realizar o experimento de fixação de voltagem ("voltage clamp") em um axônio, onde são introduzidos dois eletrodos, um responsável por medir o potencial interno e o outro por injetar a corrente elétrica. O potencial de membrana é medido pelo amplificador de voltagem (A_V) que é conectado ao eletrodo intracelular e ao terra do sistema, conectado ao banho (meio externo). O sinal do potencial de membrana (Vm) é mostrado no osciloscópio e é alimentado no terminal do amplificador de retorno (AFB). Este amplificador tem duas entradas, uma para o potencial de membrana e o outro para o potencial de comando (CP), que é um sinal elétrico escolhido pelo experimentador e que pode ter qualquer amplitude ou formato. O amplificador de retorno subtrai o potencial de membrana do potencial de comando e amplifica esta diferença, injetando uma corrente no eletrodo de corrente, mantendo o potencial interno sempre muito próximo a CP. (Kandel et al., 1991)

A corrente total da membrana I_m é a soma das correntes dos diversos íons mais a corrente capacitiva:

$$I_m(t) = I_{íons}(t) + C_m \frac{dV(t)}{dt}$$

Além do "voltage clamp", o uso de agentes farmacológicos, principalmente tetrodotoxina (TTX) e tetraetilamônio (TEA) bloqueadores de canais iônicos de sódio e potássio, respectivamente, permitiu que HH separassem a corrente iônica total em suas componentes. Assim após um grande número de experimentos, Hodgkin e Huxley postularam o seguinte modelo para a geração do potencial de ação no axônio gigante da lula (Hodgkin & Huxley 1952, Koch & Segev, 1998):

1. O potencial de ação envolve duas condutâncias iônicas principais da membrana do axônio, uma condutância de sódio, G_{Na}, e uma condutância de potássio, G_K. Estas condutâncias são independentes e o valor de cada uma delas depende de modo não linear do tempo e do potencial de membrana. Há também uma terceira condutância, bem menor que as duas anteriores, chamada condutância de "fuga", G_m, que é independente do

potencial de membrana. A corrente iônica que flui pela membrana é a soma das correntes devido a essas condutâncias:

$$I_{ions}(t) = I_{Na}(t) + I_K(t) + I_{fuga}$$

2. Cada corrente iônica individual $I_i(t)$ é linearmente relacionada (lei de Ohm) com o saldo de potencial elétrico após considerarmos o potencial de Nernst para o íon i:

$$I_i(t) = G_i(V(t),t) * (V(t) - E_i)$$

onde o potencial iônico reverso E_i é dado pela equação de Nernst para o íon específico. Dependendo do balanço entre a diferença de concentração dos íons e do campo elétrico através da membrana, cada espécie iônica tem uma “bateria" iônica associada. Conceitualmente, podemos usar o circuito equivalente mostrado na Figura 6 para descrever a membrana axonal.

3. Cada uma das duas condutâncias iônicas principais é expressa como uma condutância máxima, $G^{máx}_{Na}$ e $G^{máx}_{K}$, multiplicada por um coeficiente numérico que representa a fração da máxima condutância que se encontra aberta no momento. Estes coeficientes são funções que HH chamaram de variáveis de ativação e inativação e estariam associados a “partículas" de ativação e inativação, uma vez que HH ainda não conheciam os canais iônicos, muito menos detalhes de sua estrutura microscópica. Todas as propriedades cinéticas do modelo encontram se na descrição do funcionamento dessas partículas hipotéticas.

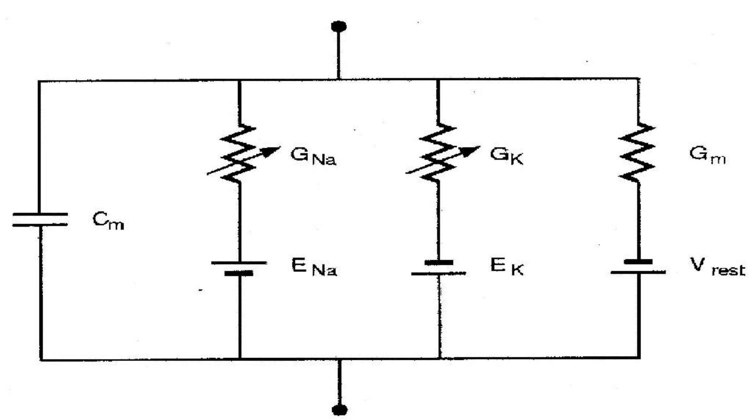


Figura 6: Circuito elétrico equivalente para o funcionamento da membrana do axônio gigante da lula. Hodgkin e Huxley elaboraram um modelo usando quatro ramos paralelos: dois passivos (a capacitância C_m e a condutância de fuga $G_m = 1/R_m$) e dois dependentes do tempo e da tensão, representando os canais de sódio e potássio.

3.1 - Corrente de Potássio I_K

A corrente de potássio (com a corrente de sódio bloqueada) foi medida através de voltage clamp e modelada por HH como:

$$I_k = G_k^{máx} * n^4 * (V - E_k),$$

onde a condutância máxima $G^{max}{}_K$= 36 mS/cm^2, e a bateria para o potássio é E_K = -12 mV , em relação ao potencial de repouso do axônio, *n* descreve o estado da partícula de ativação fictícia e é uma grandeza adimensional que assume valores entre 0 e 1. De acordo com a convenção adotada atualmente em fisiologia (corrente positiva é a que sai do neurônio), I_K é uma corrente positiva para V > E_K. HH explicaram o expoente 4 supondo que para o potássio existiam 4 partículas de ativação que deviam estar simultaneamente ativadas, uma previsão surpreendente, já que atualmente sabemos que um canal iônico de potássio é uma proteína formada por quatro subunidades idênticas, e essas quatro unidades têm que estar "abertas" para permitir o fluxo de íons.

Um modo intuitivo de entender o significado de n é considerá-lo como a probabilidade de encontrar uma partícula de ativação no estado permissivo (aberto) em que os íons podem passar e uma probabilidade 1 - n de encontrá-la no estado não permissivo (fechado), em que a passagem de íons é impedida. Se assumirmos que apenas estes dois estados existem, (para uma partícula única) e que a transição de um ao outro é governada por uma cinética de primeira ordem, ela deve obedecer ao seguinte esquema:

$$n \underset{\alpha_n}{\overset{\beta_n}{\rightleftarrows}} 1 - n,$$

α_n (β_n) é a taxa, dependente de V_m, com que ocorrem as transições do estado fechado para o aberto (aberto para o fechado) em unidades de 1/s. Este esquema corresponde matematicamente a equação diferencial de primeira ordem:

$$\frac{dn}{dt} = a_n(v) * (1 - n) - \beta_n(v)_n$$

A chave do modelo HH, e parte mais árdua de seu trabalho de modelagem, foi descrever quantitativamente a dependência em V das taxas de transição. Embora HH

tenham feito toda a formulação do modelo usando as taxas de transição α_n e β_n , uma maneira alternativa, que tem uma interpretação física mais simples, é reescrever a equação diferencial para n em termos de $\delta_n(V)$, uma constante de tempo dependente da voltagem e de $n_\infty(V)$; o valor "assintótico" de n; o valor que n assumiria p/ $t \to \infty$ mantendo-se V constante:

$$\frac{dn}{dt} = \frac{n_\infty - n}{\tau_n}$$

$$\tau_n = \frac{1}{\alpha_n + \beta_n}$$

$$n_\infty = \frac{\alpha_n}{\alpha_n - \beta_n}$$

Nesta descrição n tende exponencialmente a n_∞ com constante de tempo δ_n. As duas descrições, em termos de taxas de transição α_n β_n ou em termos de δ_n e n_∞, são equivalentes.

Ao estudar o comportamento das condutâncias individuais, HH perceberam que a relação entre a condutância e o potencial de membrana apresenta uma transição brusca: abaixo de 20 mV, G_K, a condutância da membrana ao potássio é multiplicada por 3 a cada variação de 5 mV do potencial de membrana; a sensibilidade de G_{Na} ,a condutância de sódio, é ainda maior: uma alteração de 4 mV do potencial de membrana é suficiente para multiplicar G_{Na} por 4. Para valores ainda mais despolarizados do potencial de membrana, ocorre uma saturação das condutâncias. HH assumiram que as taxas de transição deviam reproduzir a sensibilidade das condutâncias ao potencial de membrana e ajustaram aos dados experimentais as funções:

$$\alpha_n(V) = \frac{10 - V}{100\left(\exp\left(\frac{10 - V}{10}\right) - 1\right)}$$

$$\beta_n(V) = 0.125 * \exp\left(\frac{-V}{80}\right)$$

onde V é a diferença entre o potencial de membrana e o potencial de repouso em mV . Na Figura 8 mostramos os gráficos de δn e n^∞ em função de V . δn apresenta

valores máximos nas proximidades do potencial de repouso enquanto é mais rápido quando nos afastamos de V = 0. n^{∞} apresenta um crescimento monotônico com V. A curva que relaciona GK com V apresenta uma transição ainda mais brusca, por causa do expoente 4. Esse comportamento é típico de quase todas as condutâncias iônicas, uma das poucas exceções é a chamada corrente retificadora anômala que é ativada com a hiperpolarização da membrana.

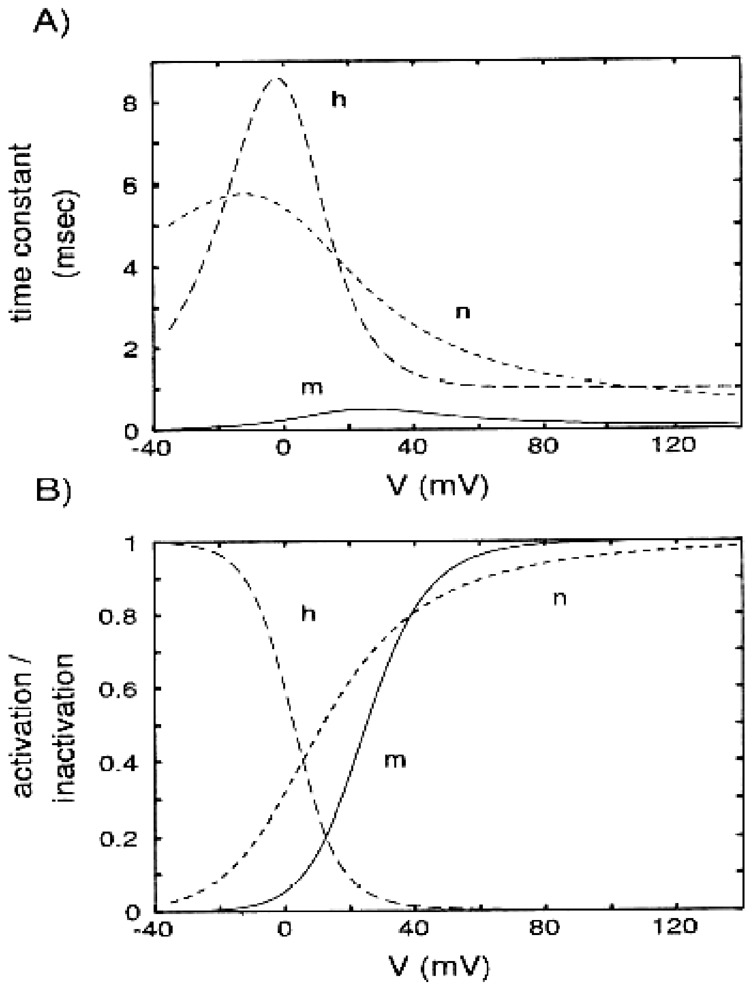


Figura 7: (A) Constantes de tempo e (B) ativação e inativação em estado estacionário em função do potencial de membrana relativo V para a ativação do sódio m (linha contínua), inativação h (linha tracejada) e ativação do potássio n (linha pontilhada).

3.2 - Corrente de Sódio I_{Na}

A dinâmica da corrente de sódio é mais complexa que a do potássio (Figura 7). Quando uma voltagem é estabelecida na membrana usando voltage-clamp a corrente de sódio aumenta com o passar do tempo, atinge um valor máximo, mas ao invés de se estabilizar como acontece com a corrente de potássio, ela volta a diminuir até parar de fluir completamente. Para ajustar este comportamento, HH postularam além da existência de uma partícula de ativação (m) para o sódio, uma partícula de inativação h:

$$I_{Na} = G_{Na_{máx}} .m^3 .h + (V - Na)$$

onde a condutância máxima $G_{Na}^{máx}$ = 120 mS/cm^2 , e a bateria para o sódio é E_{Na} = 115mV (em relação ao potencial de repouso do axônio). m e h são números adimensionais entre 0 e 1. Pela convenção que adotamos a corrente de sódio é negativa(íons Na^+ entram no axônio) na faixa de relevância fisiológica ($V < E_{Na}$).

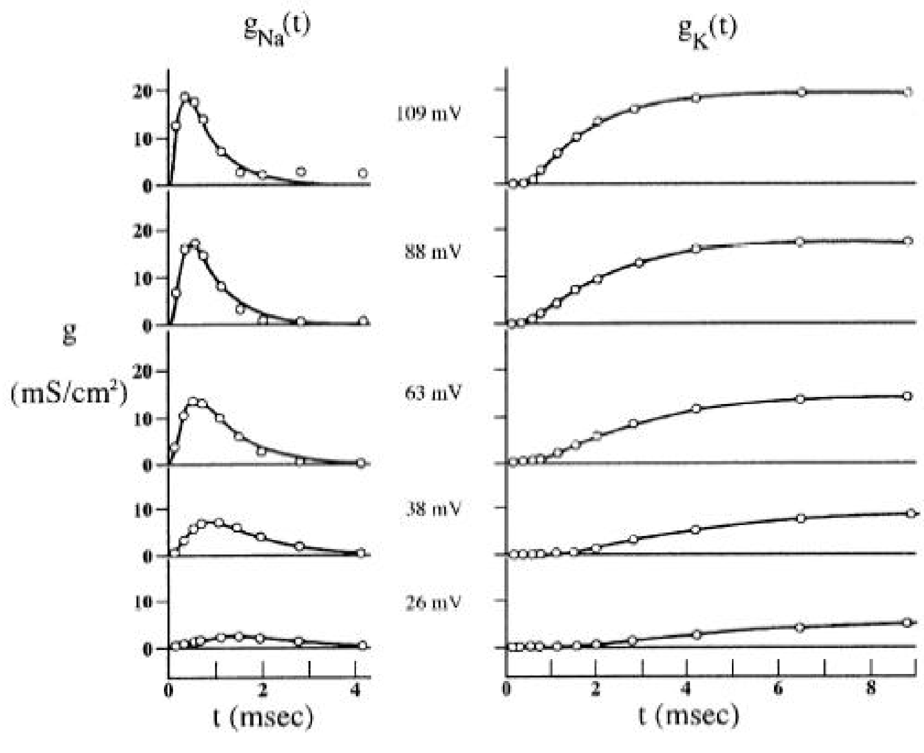


Figura 8: Os dados experimentais (círculos) e os valores calculados (linhas) de G_{Na} e G_K no neurônio gigante da lula a 6,3^0 C durante pulsos de despolarização em relação ao potencial de repouso aplicados usando voltage clamp. Para grandes mudanças de potencial, G_{Na} rapidamente cresce e depois decai devido à inativação, enquanto G_K permanece ativado. Retirado de Hodgkin e Huxley (1958).

A amplitude da condutância de sódio é determinada pelo estado de quatro partículas hipotéticas e suas transições entre os estados aberto ou fechado. Como as partículas são independentes, a probabilidade de encontrar três partículas m e uma h abertas é m^3h. Novamente, sem nenhuma informação sobre a estrutura dos canais iônicos de sódio, HH acertaram o número de subunidades de ativação e inativação que compõe um canal de sódio.

A dinâmica das partículas m e h foi modelada por HH por duas equações diferenciais de primeira ordem:

$$\frac{dm}{dt} = \alpha_m(V).(1-m) - \beta_m(V).m$$

$$\frac{dh}{dt} = \alpha_h(V).(1-h) - \beta_h(V).h$$

As taxas de transição α e β, obtidas empiricamente, são:

$$\alpha_m(V) = \frac{25-V}{100\left(\exp\left(\frac{25-V}{10}\right)-1\right)}$$

$$\beta_m(V) = 4.\exp\left(\frac{-V}{18}\right)$$

$$\alpha_h(V) = 0.07 * \exp\left(\frac{-V}{20}\right)$$

$$\beta_h(V) = \frac{1}{\exp\left(\frac{30-V}{10}\right)+1}$$

O comportamento das constantes de tempo (δ_m,δ_h) e os valores assintótico das variáveis de ativação e inativação (m^{∞} ,h^{∞}) em função de V são mostrados na Figura 8. δ_m e δ_h possuem o formato de sino, característico destas constantes de tempo, m^{∞} cresce monotonicamente com V (característica das partículas de ativação) enquanto h^{∞} decresce monotonicamente, como deve ocorrer para partículas de inativação.

3.3 - Corrente de fuga e o modelo completo para o axônio

Como ocorre com outras membranas biológicas, o axônio gigante contém uma condutância de “fuga” G_m, constante no tempo e independente de V. O valor medido por HH foi: G_m = 0,3 mS/cm2. Essa condutância passiva também tem um potencial reverso (E_r) associado a ela. HH não mediram explicitamente E_r mas ajustaram-no para que a corrente total fosse nula para V = 0. Assim, o valor E_r = 10.6 mV foi obtido por:

$$G_{Na}(0)E_{Na} + G_K(0)E_K + G_m E_r = 0$$

C_m, a capacitância da membrana para o axônio, vale 1μF/cm^2 e a resistência efetiva da membrana, no potencial de repouso devida à condutância de fuga e à pequena condutância ao sódio é 857Ω/cm^2, equivalendo a uma constante de tempo efetiva da membrana “passiva” de 0.85 ms (o que é da ordem do tempo de duração de um potencial de ação).

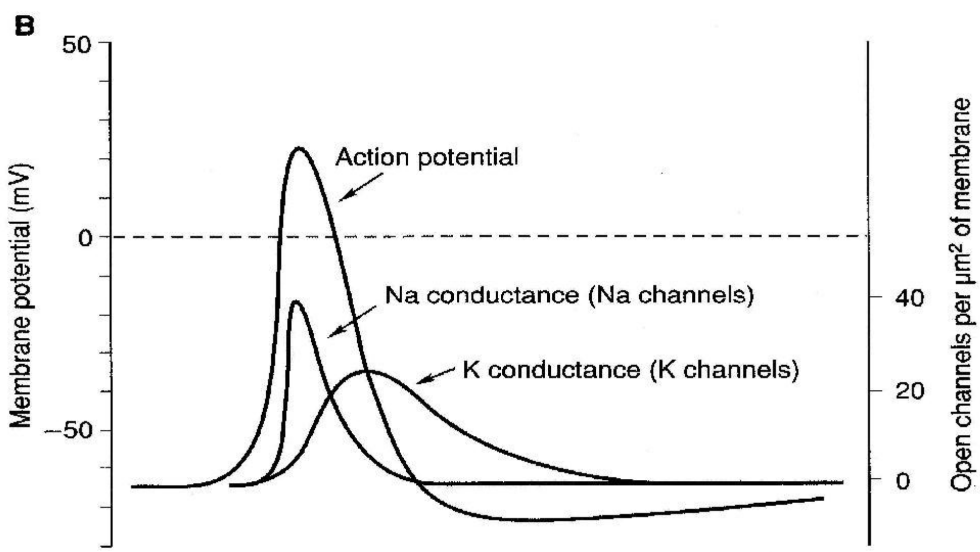


Figura 9: O potencial de ação juntamente com os valores das condutâncias dos canais de sódio e potássio. A forma do potencial de ação é determinada pelas modificações nas condutâncias dos canais de sódio e potássio.

Podemos escrever uma única equação para as correntes que fluem através de uma secção da membrana axonal:

$$C_m \frac{dV}{dt} = G_{Na}^{max} m^3 h(E_{Na} - V) + G_K^{max} n^4 (E_K - V) + G_m (E_r - V) + I_{ext}(t)$$

onde $I_{ext}(t)$ é uma corrente externa, injetada por um eletrodo inserido no axônio. Esta equação diferencial não-linear mais as três equações diferenciais lineares de primeira ordem que especificam a evolução das taxas de transição e sua dependência em V constituem o modelo clássico quadridimensional de HH para uma secção do axônio gigante da lula sob “space clamp”.

Este modelo é capaz de produzir potenciais de ação idênticos aos do axônio da lula quando um pulso de corrente externo é injetado Figura 9 e também a curva de freqüência de disparos em função de uma corrente fixa injetada.

Se a membrana celular for apenas levemente perturbada com a injeção de uma pequena corrente, ocorre que o potencial de membrana sofre uma pequena variação, mas logo tende a voltar para o potencial de repouso. No entanto, se a membrana é suficientemente excitada ($I_{ext} > 0$) ou inibida ($I_{ext} < 0$), é desencadeado um processo regenerativo que rapidamente despolariza mais ainda a membrana até inverter sua polarização e em seguida retornar a membrana ao seu potencial de repouso. A produção de potenciais de ação como resposta a uma inibição é chamada de “efeito rebote”.

No modelo HH existe um limiar do potencial de membrana (que não é fixo durante todo o funcionamento do neurônio), se esse limiar for ultrapassado durante uma perturbação acorre o disparo de um potencial de ação (Dayan & Abbot, 2001), e se não for ultrapassado, o potencial de membrana simplesmente retorna assintoticamente ao potencial de repouso. Este limiar é chamado de potencial de disparo.

3.4 - Modelo tipo HH de um neurônio do gânglio estomatogástrico de um crustáceo e suas limitações

Apesar do grande sucesso em reproduzir o potencial de ação dos neurônios obtido pelo modelo clássico HH para a membrana axonal e sua grande popularidade entre os neurocientistas, principalmente devido à correspondência direta entre os parâmetros do modelo e valores que podem ser obtidos em experimentos, há limitações em sua utilidade.

A principal limitação do modelo é que, como ele é baseado em experimentos usando "space clamp" do potencial de membrana, as flutuações espaciais do potencial de membrana e as diferentes densidades de canais iônicos não são levadas em conta. O resultado é um comportamento puramente determinístico: dada uma condição inicial e uma corrente injetada $I_{ext}(t)$; a seqüência de potenciais de ação produzida é completamente determinada e será sempre a mesma, desde que a condição inicial e $I_{ext}(t)$ sejam repetidos. Isso é bem diferente do que ocorre com axônios biológicos isolados submetidos a um sinal $I_{ext}(t)$ (Mainen & Sejnowski, 1995), conforme mostrado na Figura 10.

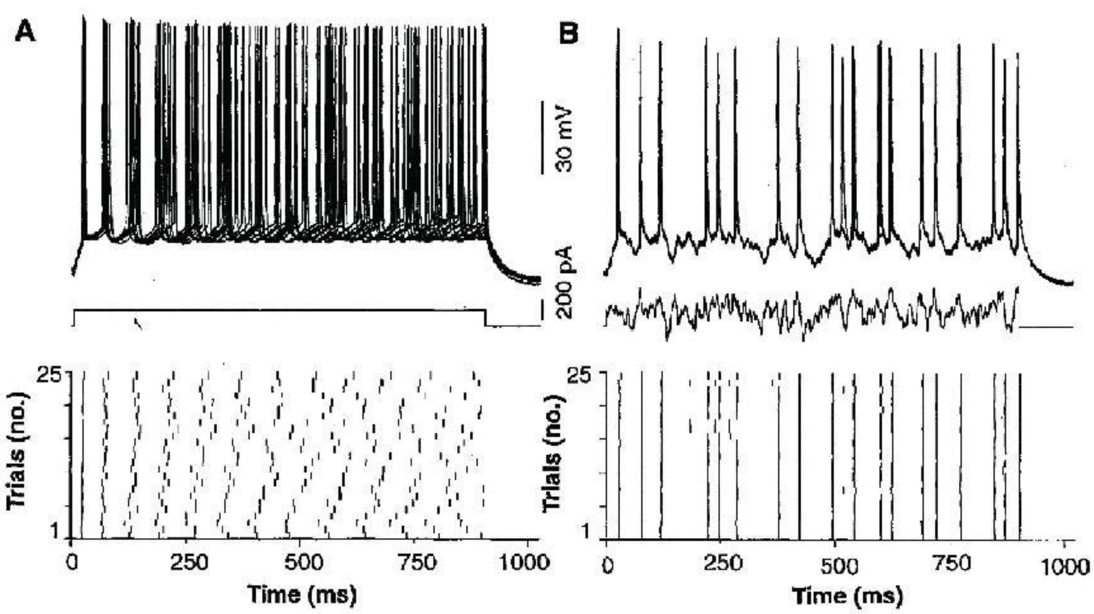


Figura 10: Reprodutibilidade do padrão de disparo de neurônios corticais causados por correntes constantes e irregulares. (A) Neste exemplo, um pulso de corrente contínua supra-liminar provocou trens de potenciais de ação irregulares, mostrado pelas diversas medidas superpostas. (B) A mesma célula foi novamente estimulada repetidamente, mas com um estímulo irregular (ruído gaussiano) e apresentou uma resposta mais regular. Retirado de Mainen e Sejnowski (1995).

Um modelo tipo HH clássico mais realista de um neurônio completo, deve ser composto de vários compartimentos. Cada compartimento isopotencial é responsável por captar a dinâmica iônica de uma região especializada do neurônio. Assim, para modelar um neurônio de modo mais acurado, são necessários vários compartimentos conectados eletricamente: um compartimento axonal similar ao modelo HH original, um compartimento somático e um ou vários compartimentos dendríticos (Falcke M, 2000).

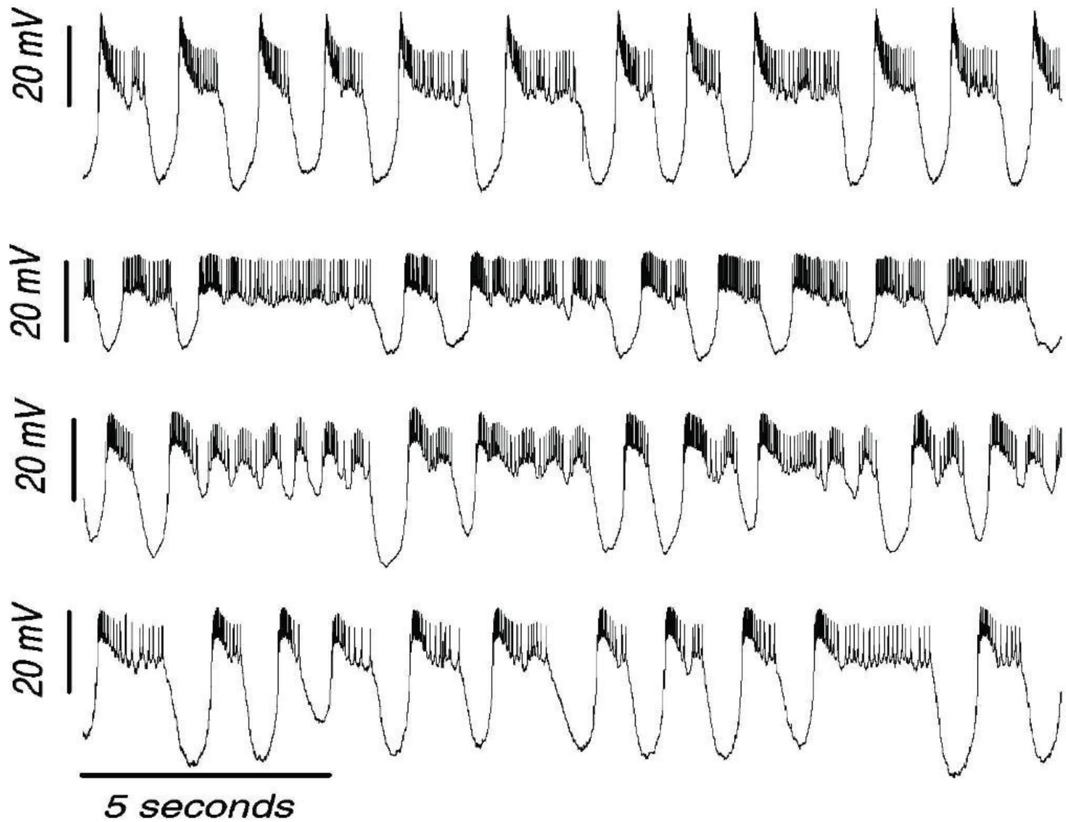


Figura 11: Séries temporais experimentais do potencial de membrana celular do neurônio lateral pilórico (LP) isolado de suas conexões sinápticas em quatro diferentes. O neurônio apresenta um padrão irregular de oscilação do potencial de membrana. Extraído de Falcke et al., 2000.

Mesmo quando não usamos vários compartimentos, o modelo precisa incluir a dinâmica de outros canais iônicos que existem na membrana neural, que produzem comportamentos mais complexos, como os mostrados na Figura 11, além dos canais responsáveis pela produção de potenciais de ação considerados no modelo original de HH. Nesses casos, mesmo um modelo de compartimento único de um neurônio “simples” de invertebrado precisa de pelo menos 7 tipos de canais iônicos diferentes, implicando na análise/simulação de 12 equações diferenciais não-lineares acopladas (Prinz , Thirumalai & Marder, 2003) e na necessidade de ajustar cada uma das funções que definem a ativação e inativação dos vários tipos de canais iônicos, conforme mostrado na Figura 12.

	p	m_∞	h_∞	τ_m	τ_h
I_{Na}	3	$\frac{1}{1+\exp\left(\frac{V+25.5}{-5.29}\right)}$	$\frac{1}{1+\exp\left(\frac{V+48.9}{5.18}\right)}$	$2.64-\frac{2.52}{1+\exp\left(\frac{V+120}{-25}\right)}$	$\frac{1.34}{1+\exp\left(\frac{V+62.9}{-10}\right)} * \left(1.5+\frac{1}{1+\exp\left(\frac{V+34.9}{3.6}\right)}\right)$
I_{CaT}	3	$\frac{1}{1+\exp\left(\frac{V+27.1}{-7.2}\right)}$	$\frac{1}{1+\exp\left(\frac{V+32.1}{5.5}\right)}$	$43.4-\frac{42.6}{1+\exp\left(\frac{V+68.1}{-20.5}\right)}$	$210-\frac{179.6}{1+\exp\left(\frac{V+55}{-16.9}\right)}$
I_{CaS}	3	$\frac{1}{1+\exp\left(\frac{V+33}{-8.1}\right)}$	$\frac{1}{1+\exp\left(\frac{V+60}{6.2}\right)}$	$2.8+\frac{14}{\exp\left(\frac{V+27}{10}\right)+\exp\left(\frac{V+70}{-13}\right)}$	$120+\frac{300}{\exp\left(\frac{V+55}{9}\right)+\exp\left(\frac{V+65}{-16}\right)}$
I_A	3	$\frac{1}{1+\exp\left(\frac{V+27.2}{-8.7}\right)}$	$\frac{1}{1+\exp\left(\frac{V+56.9}{4.9}\right)}$	$23.2-\frac{20.8}{1+\exp\left(\frac{V+32.9}{-15.2}\right)}$	$77.2-\frac{58.4}{1+\exp\left(\frac{V+38.9}{-26.5}\right)}$
I_{KCa}	4	$\frac{[Ca]}{[Ca]+3} * \frac{1}{1+\exp\left(\frac{V+28.3}{-12.6}\right)}$		$180.6-\frac{150.2}{1+\exp\left(\frac{V+46}{-22.7}\right)}$	
I_{Kd}	4	$\frac{1}{1+\exp\left(\frac{V+12.3}{-11.8}\right)}$		$14.4-\frac{12.8}{1+\exp\left(\frac{V+28.3}{-19.2}\right)}$	
I_H	1	$\frac{1}{1+\exp\left(\frac{V+75}{5.5}\right)}$		$\frac{2}{\exp\left(\frac{V+169.7}{-11.6}\right)+\exp\left(\frac{V-26.7}{14.3}\right)}$	

Figura 12: Correntes iônicas de um modelo baseado em condutâncias clássicas tipo HH para um neurônio de crustáceo. Extraído de Prinz et al. (2003).

Um modelo complexo como este apresenta vários tipos de comportamento periódico observados nos neurônios biológicos e já representa um desafio computacional, principalmente se tentarmos acoplar vários desses neurônios em uma simulação de rede. Apesar do grande número de equações não-lineares, o modelo não reproduz as oscilações irregulares encontradas nos neurônios biológicos (Figura 11) quando são utilizados valores de parâmetros numéricos biologicamente plausíveis, i.e., coerentes com os valores das condutâncias e outras grandezas obtidas experimentalmente.

Novamente, o comportamento irregular é suprimido pela simplificação determinística e pela ausência da dependência espacial tanto do valor do potencial de membrana quanto da distribuição de canais iônicos.

4. Modelo Dinâmico do Tipo Hindmarsh-Rose (HR) e Neurônios Eletrônicos

Estamos chamando de modelagem dinâmica as abordagens que consistem em modelar o comportamento dinâmico do potencial de membrana, sem levar em conta a contribuição individual de diferentes grupos iônicos, ou compartimentos, ou utilizar ferramentas e/ou argumentos da teoria de sistemas dinâmicos para auxiliar na modelagem de um comportamento desejado.

Diversas simplificações podem ser úteis para construir modelos com um número reduzido de equações diferenciais e possibilitar a simulação computacional de redes neurais compostas por vários neurônios.

Um modelo dinâmico bastante interessante da produção de potenciais de ação, que usa apenas duas equações diferenciais de primeira ordem, foi proposto por Hindmarsh e Rose (H-R) em 1982 (Hindmarsh & Rose, 1982). O modelo H-R pode ser interpretado como uma generalização das equações de Fitzhugh (Fitzhugh, 1961), mas pode ser desenvolvido partindo de princípios básicos se assumirmos que a taxa de mudança do potencial de membrana depende linearmente de z, uma corrente introduzida através de um eletrodo, e de y, uma corrente intrínseca da membrana, e depende de maneira não linear do próprio potencial de membrana:

$$\frac{dx}{dt} = -a(f(x) - y - z)$$

H-R também supôs que a taxa de variação de y é dada por:

$$\frac{dy}{dt} = b(g(x) - y)$$

Nestas equações, a e b são constantes e a equação para dy/dt tem este formato para garantir que a evolução temporal de z em condições de “voltage-clamp” seja exponencial (Hindmarsh & Rose, 1982).

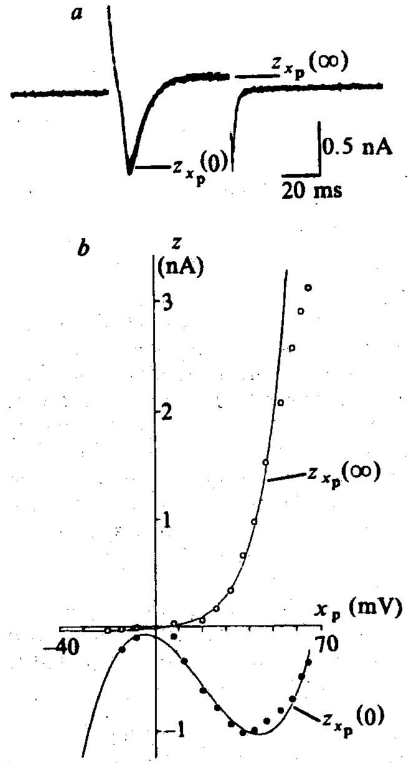


Figura 13: (a) Corrente z necessária para produzir um degrau de voltagem (xp) de 25 mV a partir de xp = 0 nos experimentos de Hindmarsh e Rose com neurônios de Limnaea stagnalis. (b) Valores das correntes zxp(0) e zxp (∞) em função do potencial fixado e respectivas funções ajustadas. Extraído de Hindmarsh e Rose,1982.

Para determinar a forma de f(x) e de g(x), H-R realizaram experimentos de ”voltage-clamp" com células do gânglio visceral do molusco Lymnaea stagnalis, conforme mostrado na Figura 13.

Para diferentes valores do potencial fixado x_p, H-R e mediram o valor da corrente $z_{xp}(0)$ necessária para manter o potencial no início do “voltage clamp" e a corrente assintótica $z_{xp}(\infty)$.

Nestas condições,

$$z_{xp}(0) = f(x_p)$$

$$z_{xp}(\infty) = f(x_p) - g(x_p)$$

e os ajustes aos dados experimentais fornecem as funções f(x) e g(x).

Usando este modelo e as funções ajustadas:

$$f(x) = cx^3 + dx^2 + ex + h$$

$$g(x) = f(x) - qe^{rx} + s$$

onde a - h, q, r e s são constantes, H-R reproduziram os potenciais de ação produzidos pelos neurônios de Lymnaea como mostrado na Figura 14.

Este modelo de apenas duas variáveis, apesar de não reproduzir detalhes do formato dos potenciais de ação, foi capaz de capturar outras propriedades como o fato dos potenciais de ação serem separados por um longo intervalo, como acontece em neurônios biológicos e não era reproduzido por outros modelos simplificados como o modelo de Fitzhugh (Fitzhugh,1961). Em 1984, H-R propôs uma adaptação de seu modelo bidimensional para permitir a reprodução do comportamento de neurônios que disparam potenciais de ação em salvas (“bursts") (Hindmarsh & Rose, 1984).

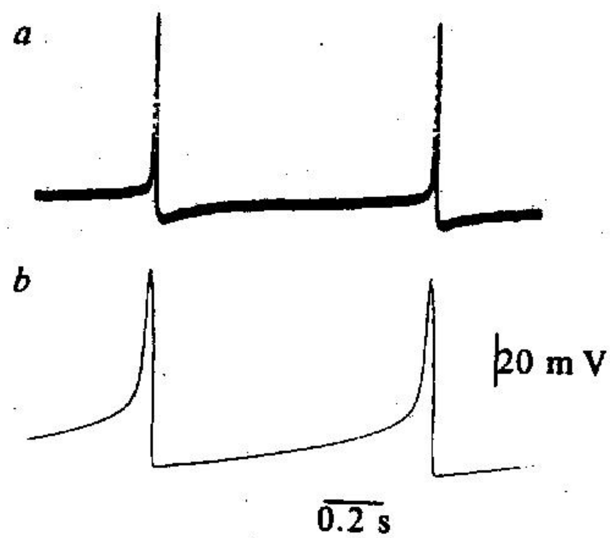


Figura 14: Comparação entre as formas dos potenciais de ação (a) medido e (b) originado pelo modelo bidimensional de Hindmarsh e Rose, 1982.

A representação do modelo original de H-R, após a mudança de variáveis: v=x, r = y, I = z, apresenta um único ponto fixo (foco instável) num espaço de fases I - v, que corresponde ao cruzamento das curvas nuliclinais: dv/dt = dx/dt = 0 e dr/dt = dy/dt = 0, conforme mostrado na Figura 15. O valor longo do intervalo entre potenciais de ação é produzido pela proximidade das nulclinas quando v < 0. Nesta região a solução se move devagar ao longo do canal entre as nulclinas, durante a hiperpolarização e rapidamente quando as nuliclinais estão afastadas, durante o potencial de ação.

A modificação do modelo para incluir o comportamento em salvas consistiu em deformar “levemente" a nuliclinal dr/dt = 0 para que ela cruzasse novamente a nuliclinal dv/dt = 0, produzindo mais dois pontos fixos no espaço de fases, conforme mostrado na Figura 15.

H-R analisaram a estabilidade dos pontos fixos e mostraram que um deles

corresponde a um ponto fixo estável para o qual a dinâmica se dirige assintóticamente, após produzir vários potenciais de ação.

Como as nuliclinais continuam sempre próximas em v < 0, H-R introduziram uma terceira equação diferencial em seu modelo, com uma dinâmica mais lenta, que corresponderia a adicionar um valor a dr/dt que pudesse chavear o sistema entre duas possibilidades: um único ponto fixo instável, correspondente a produção seguida de potenciais de ação, e três pontos fixos, sendo um deles estável, para o qual o sistema se dirige assintóticamente, correspondendo a uma hiperpolarização ou relaxamento, conforme mostrado na Figura 16.

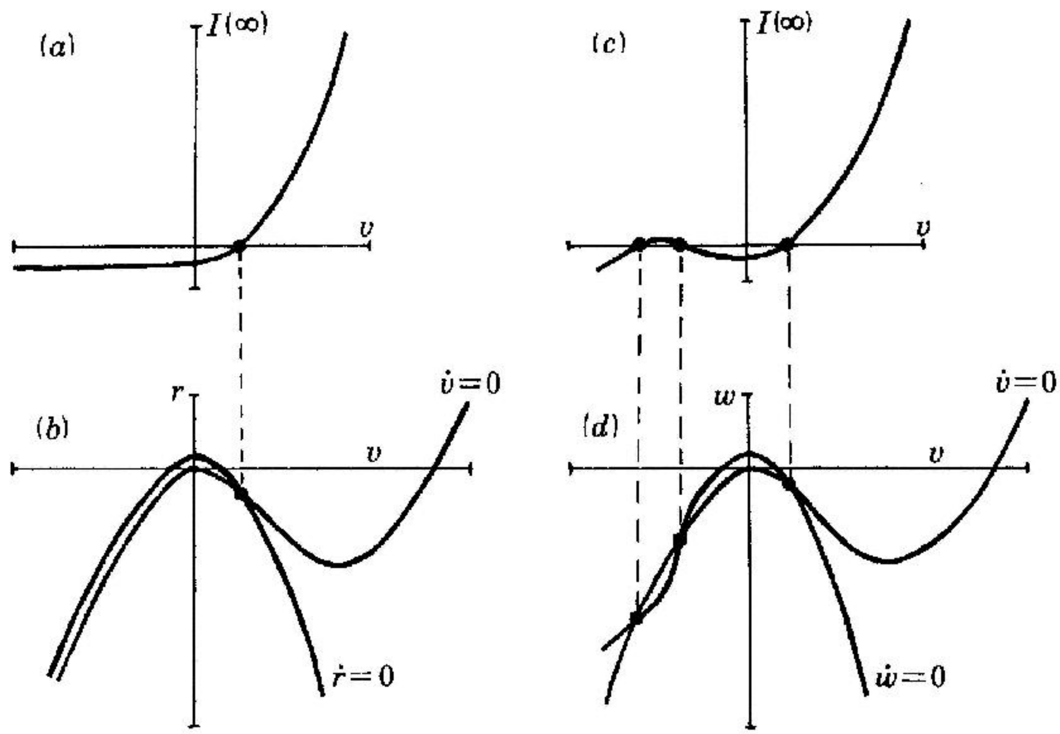


Figura 15: Curvas I - v do modelo H-R bidimensional: (a) para o modelo com um único ponto fixo, (c) para o modelo com três pontos fixos; os respectivos espaços de fase (r ¡ v) em (b) e (d), mostrando o cruzamento das nuliclinais de r e v.

O modelo de H-R tridimensional que apresenta salvas de potenciais de ação é dado por:

$$\frac{dx}{dt} = y + ax^3 + bx^2 + I - z \quad + I_{syn}$$

$$\frac{dy}{dt} = c - dx^2 - y$$

$$\frac{dz}{dt} = r(s(x - x_1) - z)$$

onde x, y, e z são as variáveis dinâmicas que representam respectivamente o potencial de membrana, uma corrente rápida e uma corrente lenta; todos os outros parâmetros são constantes. I_{syn} é uma corrente externa que pode ser usada para simular a soma de todas a correntes sinápticas de entrada do neurônio.

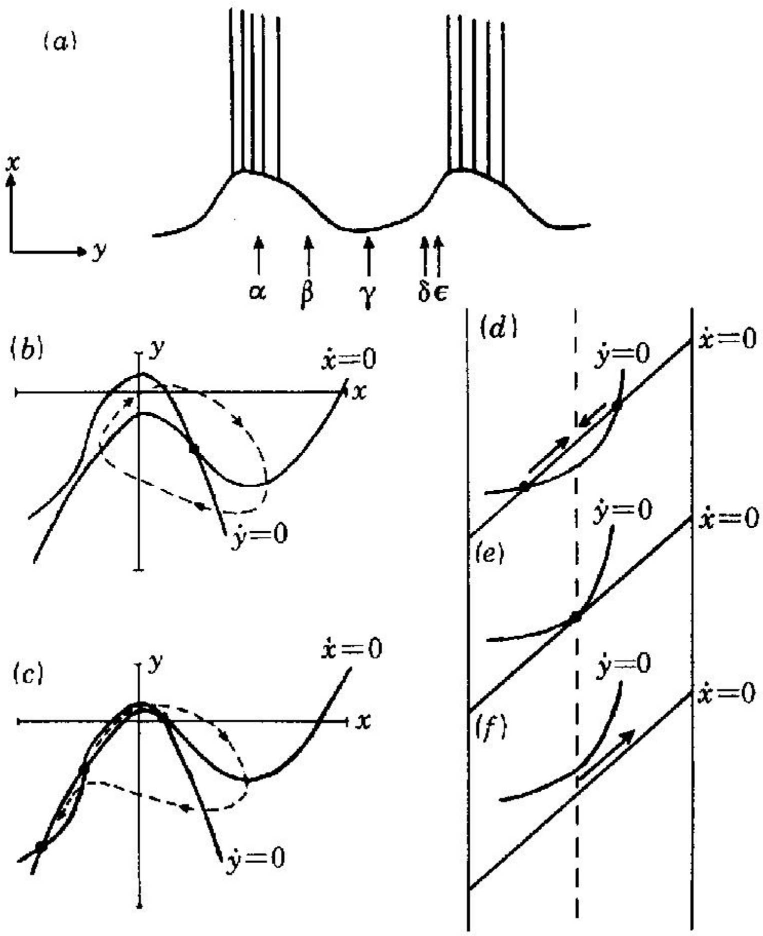


Figura 16: (a) Representação esquemática do ciclo de uma salva de potenciais de ação no modelo H-R; (b) e (c) são diagramas de fase do plano xy que correspondem aos tempos assinalados como α e β em (a); (d) a (f) mostram detalhes do plano de fase xy que correspondem respectivamente aos tempos γ a ε do ciclo da salva.

Uma característica notável deste modelo simplificado, com apenas 3 equações diferenciais, foi sua capacidade não apenas de apresentar salvas de potenciais de ação, mas também sensibilidade às condições iniciais que produzem salvas de potenciais de ação com tempos de duração irregulares como os encontrados em neurônios biológicos.

Baseado-se nestas propriedades de produzir salvas irregulares, este modelo foi adaptado com a adição de mais uma equação diferencial para tornar seu comportamento ainda mais parecido com os neurônios biológicos do gânglio estomatogástrico de crustáceos (Pinto et al., 2000).

O modelo H-R adaptado aos crustáceos reproduziu qualitativamente os mesmos tipos de oscilação irregular encontrados nos neurônios biológicos, conforme pode ser visto comparando a Figura 17 com a Figura 11.

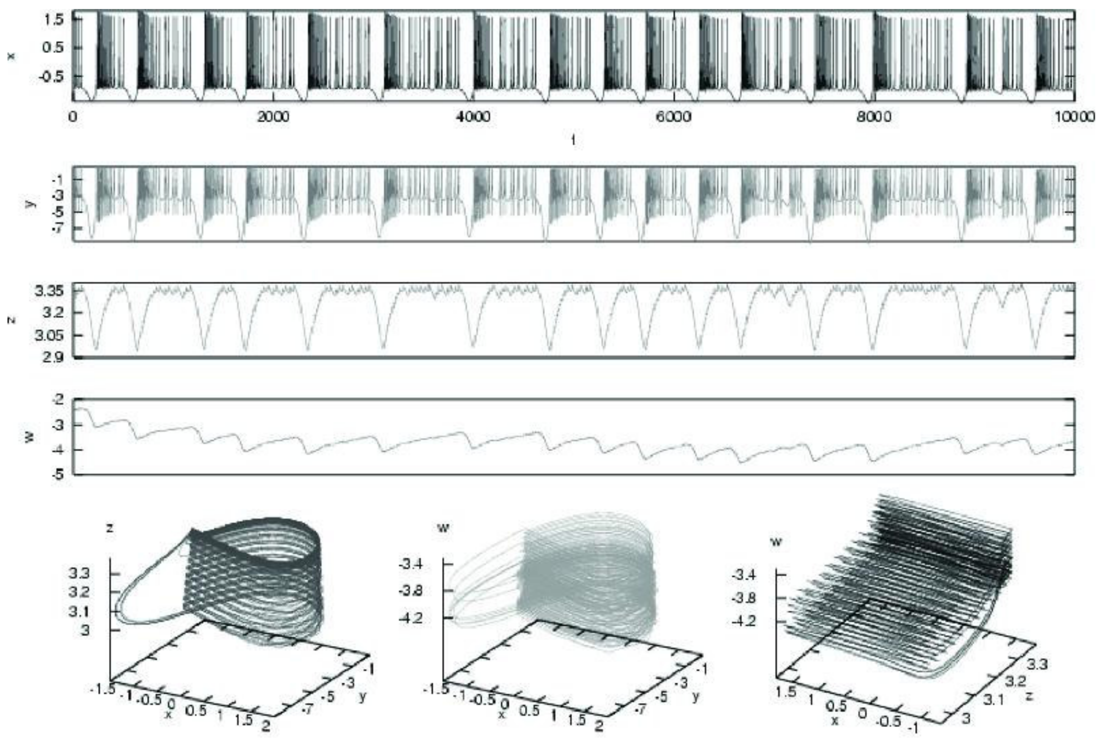


Figura 17: Séries temporais obtidas com o modelo H-R implementado em circuito eletrônico (superior) e projeções tridimensionais do atrator dinâmico caótico. (Pinto et al., 2000)

Como as equações diferenciais que compõe o modelo H-R são simples, envolvendo apenas polinômios em x, elas podem ser simuladas utilizando circuitos eletrônicos analógicos formados por multiplicadores e divisores de tensão e integradores. Um modelo deste tipo pode ser chamado de computador analógico ou de “neurônio eletrônico" (EN). ENs baseados no modelo H-R (Pinto et al., 2000) foram utilizados com sucesso para restaurar o funcionamento de um circuito neural biológico danificado (Szücs et al., 2000, Selverston et al., 2000).

4.1 - Limitações de modelos tipo HR

Muito embora um único EN conectado a neurônios biológicos ou ENs conectados aos pares funcionem de modo muito similar a neurônios biológicos, eles não são capazes de reproduzir o funcionamento de uma pequena rede neural, composta por três quatro neurônios, quando usamos apenas ENs. Isso acontece porque, apesar de apresentarem salvas de duração bastante variável, as hiperpolarizações, que definem a distância entre salvas, são muito parecidas e não permitem a mesma flexibilidade apresentada pelos neurônios biológicos.

Além dessa limitação, este modelo também é definido por equações diferenciais determinísticas, ou seja, não apresenta variabilidade na produção de potenciais de ação quando submetido repetidamente a um mesmo sinal.

5. Modelo de Hodgkin-Huxley Estocástico

No modelo clássico HH a variação dos valores das diversas condutâncias da membrana são considerados como processos contínuos e determinísticos. Entretanto, os processos de perfusão de íons através da membrana não são nem contínuos, nem determinísticos (Koch & Segev, 1998). Canais iônicos são elementos discretos com propriedades que só podem ser dadas de maneira probabilística.

Como a membrana axonal é constituída por um número muito grande de canais iônicos, da ordem de até centenas de milhares para o caso do sódio (Hille, 2001), acreditou se por muito tempo que o axônio tenderia a se comportar de modo determinístico, usando uma espécie de “média" do comportamento de todos os canais, justificando a utilização de modelos determinísticos.

Vários trabalhos experimentais, como o de Mainen & Sejnowski (Mainen & Sejnowski, 1995), tem mostrado que o potencial de membrana macroscópico dos neurônios possuí características que não podem ser explicadas pelos modelos determinísticos da produção de potenciais de ação.

A justificativa para usar um modelo estocástico para a dinâmica dos canais iônicos vem do fato que, em condições normais e bem diferentes daquelas em que o axônio inteiro se encontra em “space clamp", os potenciais de ação são iniciados em uma região bem pequena do axônio, chamada cone de implantação (“axon hilock"). Esta região tem, tipicamente, 200 μm^2 e possui aproximadamente 3500 canais seletivos a potássio e uns 12000 canais seletivos a sódio (Schneidman, Freedman & Segev, 1998). Se considerarmos que o número de canais que se comportam de modo diferente da média seja da ordem de $\frac{\sqrt{n}}{n}$ canais (lei dos grandes números), teríamos uma flutuação de aproximadamente 1%, que pode ser muito importante quando consideramos um sistema dinâmico não linear como a membrana neuronal.

A modelagem estocástica parte dos mesmos princípios utilizados pelo modelo HH clássico. HH sugeriram um interpretação física para os expoentes que ajustaram a suas equações, dizendo que o termo n^4 indica que existem 4 portas independentes e um canal de potássio só estaria aberto se todas as portas estivesses abertas. Este é exatamente o caso, conforme foi descoberto mais tarde usando técnicas de biologia molecular: um canal seletivo a K^+ possui quatro estruturas independentes e apenas um estado que

corresponde a todas as estruturas em uma conformação aberta permite o fluxo de íons através do canal, conforme mostrado na Figura 18.

Analogamente, o termo m^3h (capitulo 3.2)reflete a existência de 3 portas m e uma porta em um canal iônico seletivo a sódio e que todas estas portas devem estar abertas para permitir a passagem de um íon Na^+.

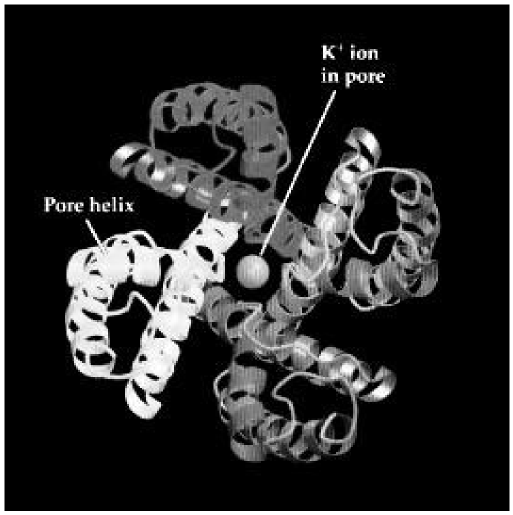


Figura 18: Representação da estrutura de um canal iônico de potássio revelando as estruturas que funcionam como portas: as quatro estruturas têm que estar na conformação espacial que corresponde a aberto para que os íons possam fluir pelo poro.

Este modelo sugere que os canais de K^+ podem existir em 5 estados diferentes, num esquema cinético que pode ser formalizado como uma cadeia Markoviana que incorpore a dinâmica das transições entre os estados, conforme mostrado na Figura 19.

$[n_i]$ representa o número de canais com i portas abertas, portanto, $[n_4]$ é o único estado que significa canais de K^+ abertos. As taxas de transição entre os estados, α_n e β_n são idênticas às taxas de transição entre as partículas de ativação originalmente propostas por HH.

Analogamente, cada canal de sódio, pode existir em 8 estados diferentes, conforme a cadeia Markoviana mostrada na Figura 20.

Neste caso, $[m_ih_j]$ é o número de canais Na^+ no estado m_ih_j e o único estado que corresponde a canais abertos é o m_3h_1. Novamente, as taxas de transição α_m e β_m, α_h e β_h são as taxas de transição do formalismo original clássico de HH.

$$[n_0] \underset{\beta_n}{\overset{4\alpha_n}{\rightleftharpoons}} [n_1] \underset{2\beta_n}{\overset{3\alpha_n}{\rightleftharpoons}} [n_2] \underset{3\beta_n}{\overset{2\alpha_n}{\rightleftharpoons}} [n_3] \underset{4\beta_n}{\overset{\alpha_n}{\rightleftharpoons}} [n_4]$$

Figura 19: Diagrama esquemático representando os cinco estados possíveis que um canal de potássio pode assumir e as taxas de transição entre cada par de estados. O índice indica o número de portas abertas do canal. O único estado que corresponde a um canal aberto é o N_4.

$$\begin{array}{ccccccc}
[m_0h_1] & \underset{\beta_m}{\overset{3\alpha_m}{\rightleftharpoons}} & [m_1h_1] & \underset{2\beta_m}{\overset{2\alpha_m}{\rightleftharpoons}} & [m_2h_1] & \underset{3\beta_m}{\overset{\alpha_m}{\rightleftharpoons}} & [m_3h_1] \\
\alpha_h \uparrow\downarrow \beta_h & & \alpha_h \uparrow\downarrow \beta_h & & \alpha_h \uparrow\downarrow \beta_h & & \alpha_h \uparrow\downarrow \beta_h \\
[m_0h_0] & \underset{\beta_m}{\overset{3\alpha_m}{\rightleftharpoons}} & [m_1h_0] & \underset{2\beta_m}{\overset{2\alpha_m}{\rightleftharpoons}} & [m_2h_0] & \underset{3\beta_m}{\overset{\alpha_m}{\rightleftharpoons}} & [m_3h_0],
\end{array}$$

Figura 20: Diagrama esquemático representando os 8 estados possíveis em que um canal de sódio pode ser encontrado e as taxas de transição α e β entre os estados. O único estado que corresponde a um canal que permite a passagem de íons é o m_3h_1

No formalismo HH estocástico as condutâncias do sódio e potássio são dadas por:

$$G_k(V,t) = \gamma k[n_4]$$

$$G_{Na}(V,t) = \gamma Na[m_3h_1]$$

onde γ_K e γ_{Na} são os valores das condutâncias de um único canal iônico de K^+ ou Na^+ no estado aberto.

Ou seja, o problema original de HH era integrar 3 equações diferenciais para obter os valores de n, m e h mais uma equação diferencial para obter V , enquanto no modelo estocástico há apenas uma equação diferencial a ser integrada. Entretanto a partir das taxas de transição temos que determinar, usando sorteios de números aleatórios, se cada canal iônico permanece no estado em que está ou muda para um estado próximo. Se considerarmos um número de canais da ordem de 10^5 teríamos que sortear 10^5 números aleatórios a cada instante para determinar a "sorte" da população total de canais iônicos, o que é até factível, mas bastante dispendioso em termos de tempo de computação.

Na verdade este problema técnico é contornado usando métodos que calculam

qual o número total de canais que mudam de um estado para outro, reduzindo bastante o tempo de computação. Se, em um dado instante de tempo t, há N_A canais no estado A, N_B canais no estado B e a taxa de transferência de canais do estado A para o estado B vale **r** no instante t, então cada um dos canais no estado A pode se transferir para o estado B no intervalo entre t e t + Δt com uma probabilidade p = rΔt. Assim, em cada passo de tempo Δt podemos determinar o número de canais que se movem de A para B, Δn_{AB}, escolhendo um número aleatório de uma distribuição de probabilidade binomial:

$$prob(\Delta r_{AB}) = \binom{n_A}{\Delta n_{AB}} p^{\Delta n_{AB}} (1-p)^{(n_A - \Delta n_{AB})}$$

O resultado obtido com o modelo HH estocástico comparado ao modelo HH determinístico original pode ser observado na Figura 21.

O modelo HH estocástico do axônio reproduz os resultados de confiabilidade e reprodutibilidade da seqüência de potenciais de ação obtidos em experimentos com neurônios biológicos quando sinais de corrente tipo degrau ou sinais com flutuação são aplicados repetidamente ao neurônio, enquanto o modelo HH determinístico do axônio repete sempre a mesma seqüência de potenciais de ação para o mesmo estímulo.

A variabilidade das respostas apresentadas pelos modelos estocásticos podem ser a chave para resolver os problemas de flexibilidade que modelos HH determinísticos e modelos dinâmicos tipo H-R apresentam. Entretanto um modelo completo do tipo HH estocástico para um neurônio inteiro, necessita de pelo menos de 2 compartimentos: uma pequena área de membrana axonal conectada eletricamente a um compartimento somático para produzir um potencial de membrana realístico. Além disso, o tempo necessário para computação depende muito do número de canais diferentes presentes no modelo, e a simulação estocástica é mais lenta que a integração do modelo HH determinístico. As desvantagens dos modelos estocásticos são, em sua maioria, de ordem técnica e devem ser solucionadas em um futuro próximo, com o aparecimento de novas tecnologias computacionais mais velozes que as atuais.

Nos testes com o modelo estocástico de um neurônio estomatogástrico descritos no Capítulo 8 utilizamos uma implementação com dois compartimentos desenvolvida por Carelli e colaboradores (Carelli et al., 2005) que consiste em uma versão estocástica do modelo determinístico original descrito na seção 3.4.

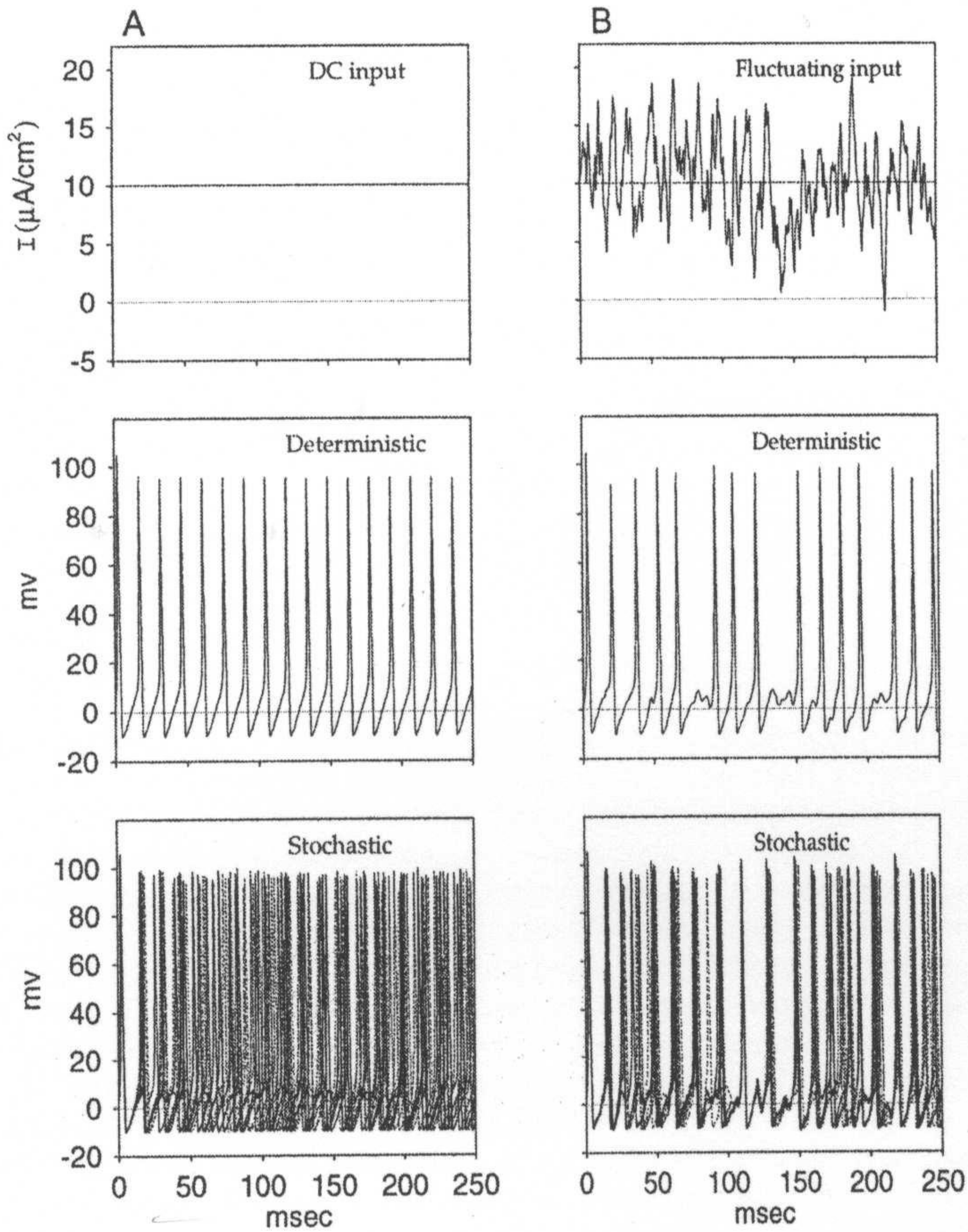


Figura 21: Comparação da confiabilidade e variabilidade dos padrões de disparos de potenciais de ação entre um modelo axonal tipo HH clássico e um modelo axonal HH estocástico; (A) representa a resposta dos modelos a um degrau de corrente contínua; (B) representa a resposta dos modelos a uma corrente que apresenta flutuações gaussianas em torno de um valor médio. Em ambos os casos, o sinal foi repetido dez vezes e as curvas obtidas para o potencial de membrana foram superpostas.

6. O Protocolo Dynamic Clamp

As principais ferramentas usadas para investigar as características de uma particular condutância em um neurônio são os métodos conhecidos como voltage-clamp e patch-clamp. Embora essenciais para a compreensão das propriedades elétricas e temporais de uma determinada condutância, estes métodos não podem ser aplicados ao estudo do comportamento coletivo de todas as condutâncias que determinam como um neurônio se comporta durante sua atividade normal, seja individualmente ou em um circuito.

A maioria dos experimentos que utilizam voltage ou patch-clamp fixam ou limitam os valores de voltagem que a membrana do neurônio em estudo pode apresentar e, normalmente, agentes farmacológicos são utilizados para isolar um único tipo de condutância. Desse modo, estes métodos clássicos não permitem, durante a operação normal do neurônio, estudar as alterações das condutâncias da membrana da célula devido às suas propriedades elétricas intrínsecas ou em resposta aos sinais recebidos pelas sinapses com outros neurônios.

O método conhecido como Dynamic Clamp consiste em utilizar um computador para introduzir condutâncias artificiais ou simular sinapses artificiais na membrana em um neurônio biológico (Sharp et al., 1992, 1993, 1996; Pinto et al., 2001; Butera et al., 2001; Dorval et al., 2001; Prinz et al., 2004). O modo como estas condutâncias dependem da voltagem da membrana ou do tempo é especificado por equações matemáticas que são resolvidas em tempo real por um computador conectado ao neurônio biológico. De certo modo, o Dynamic Clamp utiliza os neurônios como simuladores, permitindo investigar a importância de um tipo de condutância para a atividade elétrica de um neurônio, assim como determinar o efeito produzido pelas sinapses em uma rede, combinando o controle e flexibilidade de uma simulação no computador com a acurácia e o realismo de um experimento em eletrofisiologia, na Figura 22 esta representado o esquema em bolcos do funcionamento do Dynamic Clamp. Retirado e adaptado de Dorval et al., (2001).

Para simular a presença de condutâncias na membrana de um neurônio ou a existência de uma sinapse entre dois neurônios, é implementado um ciclo onde:

(i) A voltagem de membrana dos neurônios é medida através de eletrodos intracelulares, amplificada adequadamente, digitalizada em um ADC e lida

pelo computador.

(ii) Baseando-se nas voltagens de membrana, o computador simula em tempo real as equações matemáticas que descrevem as condutâncias e sinapses, calculando as correntes a serem injetadas nos neurônios devido à composição de todas as condutâncias artificiais que estão sendo geradas.

(iii) Através de um DAC são produzidas tensões de saída proporcionais às correntes a serem injetadas nos neurônios através de microeletrodos (após tratamento adequado nos amplificadores).

Esse ciclo é repetido durante todo o processo e, assim, as condutâncias vão sendo alteradas dinamicamente de acordo com o valor da voltagem de membrana do próprio neurônio ou com o sinal vindo de um outro neurônio (sinapse artificial). Desse modo, o Dynamic Clamp produz o efeito elétrico das condutâncias e sinapses artificiais como se todas elas estivessem localizadas na ponta do microeletrodo de corrente.

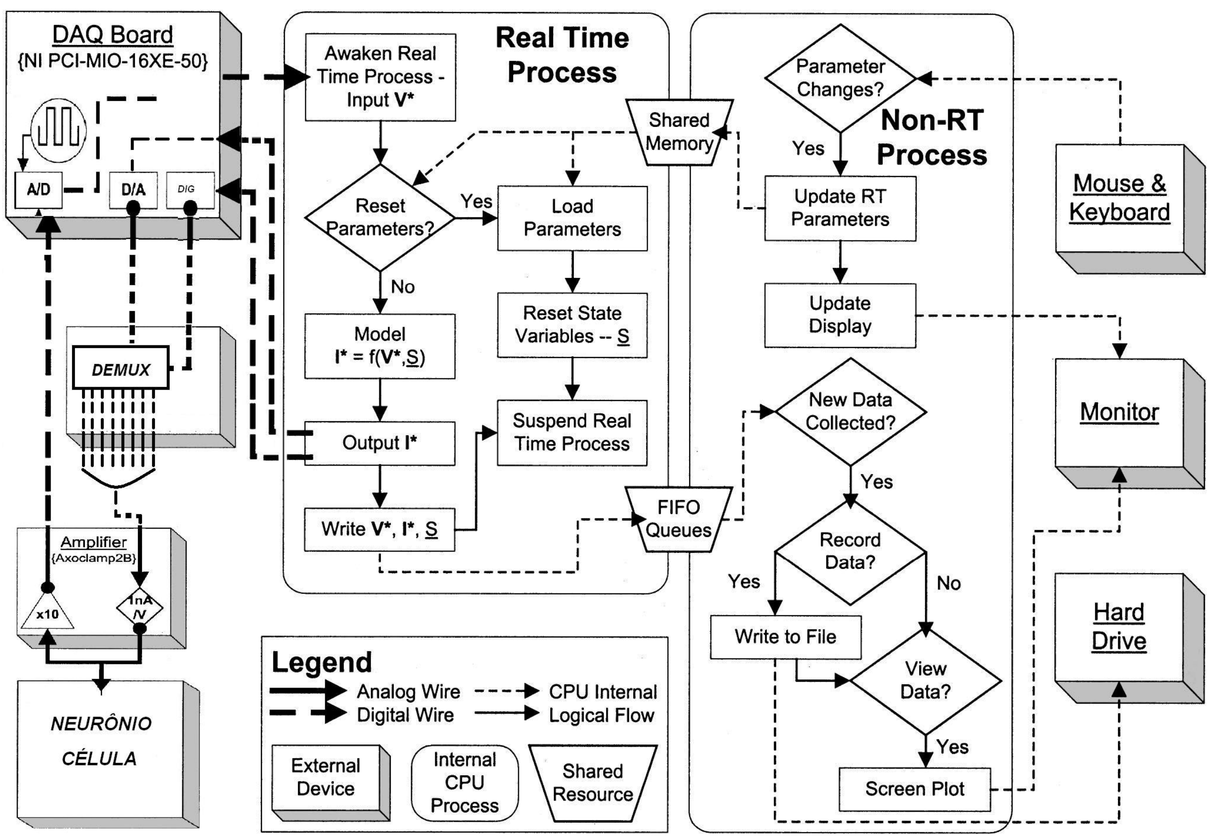


Figura 22: Diagrama em blocos. Amplificador lê o potencial de membrana analógico do eletrodo da célula (Neurônio). O DAQ amostra esta voltagem, converte o sinal analógico para digital passando para um processo RT a uma freqüência fixa. O RT utiliza os valores da voltagem, e através de um modelo de condutância calcular uma corrente, junto com a corrente de saída é gerado um sinal digital de controle para demultiplexação, sendo essa corrente amplificada adequadamente e injetando o sinal na célula. Retirado e adaptado de Dorval et al., (2001).

6.1 - Sinapses artificiais

Sinapses químicas

O modelo utilizado para calcular a corrente a ser injetada em um neurônio (célula pós-sináptica) nas sinapses químicas é baseado nas equações que descrevem a cinética de primeira ordem (Destexhe et al.,1994).

$$I_{\sin} = g_{\sin}.S.(V_{\sin} - V_{pos}) \qquad (1)$$

$$(1 - S_\infty).\tau_S \frac{dS}{dt} = (S_\infty - S) \qquad (2)$$

Onde ,

$$S_\infty(V_{pre}) = \begin{cases} tgh\left[(V_{pre} - V_{th})/V_{inc}\right] se V_{pre} > V_{tr} \\ 0, se V_{pre} < V_{tr} \end{cases}$$

g_{sin} é a máxima condutância sináptica, S a ativação instantânea , S_∞ a ativação final, V_{sin} o potencial reverso da sinapse, V_{pre} e V_{pos} são as voltagens de membrana das células pré e pós-sináptica, respectivamente, τ_S é a constante de tempo da sinapse , V_{th} é o potencial de limiar de ativação e V_{inc} é o potencial que define a inclinação da curva de ativação da curva de ativação para S_∞.

Integrando-se algebricamente a equação diferencial em (2) obtemos

$$S(\Delta t) = S_\infty + C.\exp.\left(-\frac{\Delta t}{(S_\infty - 1).\tau_s}\right), \qquad (3)$$

onde C é uma constante de normalização que fica definida quando impusermos que S(t) obedeça às condições iniciais conforme descrevemos abaixo.

A cada ciclo de Dynamic Clamp os valores obtidos para a voltagem de membrana dos neurônios produzem um valor de $S\infty$. Supondo que o ciclo é repedido após um intervalo de tempo t (que pode ser medido com precisão de 250ns acrescentando-se o relógio da placa de aquisição), podemos calcular como a ativação instantânea da sinapse descrita por (2) teria evoluído após este intervalo de tempo, ou seja, determinamos a constante C através de :

$$S_{ant} = S(0) = S_{\infty ant} + C.\exp.(0) \Rightarrow C = (S_{ant} - S_{\infty ant}) \qquad (4)$$

Onde S_{ant} é a ativação instantânea no instante prévio e $S_{\infty ant}$ é o valor da ativação final no instante prévio. A ativação instantânea em t é, então, dada por

$$S(\Delta t) = S_{\infty ant} + (S_{ant} - S_{\infty ant}).\exp.\left(-\frac{\Delta t}{(S_{\infty} - 1).\tau_s} \right) \qquad (5)$$

Com esses valores discretos da ativação instantânea, obtidos a cada ciclo de Dynamic Clamp calculamos a corrente a ser injetada no neurônio de acordo com (2). Como o ciclo é repetido a uma freqüência relativamente alta (~10 kHz), a corrente injetada no neurônio tem uma forma suave e continua.

6.2 - Implementações do Dynamic Clamp e limitações

Com o objetivo de atingir maiores taxas de atualização e acompanhar a evolução do hardware (microprocessadores mais rápidos) várias implementações de Dynamic Clamp se sucederam desde a original de Sharp e colaboradores:

- DClamp (sistema operacional DOS e microprocessador<Pentium) (Sharp et al., 1992,1993,1996);
- DynClamp4 (sistema operacional Windows NT/95/98/2000/XP) (Pinto et al., 2001) taxa de atualização de até 10 KHz, possue quatro canais de saída;
- MRCI (sistema operacional Linux, software driver caseiro, tempo real) (Butera et al., 2001) taxa de atualização de até 30 KHz, possue dois canais de saida;
- Real Time Linux Dynamic Clamp – RTLDC (sistema Linux, software driver *COMEDI* com código aberto, tempo real) (Dorval et al., 2001) taxa de atualização de até 20 KHz, possue dois canais de saída;

As aplicações do Dynamic Clamp e plataforma Windowns tem uma taxa de atualização de 2 a 20 KHz dependendo do modelo de condutância que esta sendo utilizado, a desvantagem do sistema Windows esta no tempo de processamento da informação, o sistema trata das interrupções ele divide o processamento entre as operações que esta realizando, variando a velocidade de atualização.

No sistema Linux as versões desenvolvidas do Dynamic Clamp em tempo real evita as interrupções que ocorrem no sistema Windows e as taxas de atualização são de 20-50 KHz dependendo da placa de aquisição de dados.

Uma comparação entre as características, qualidades e problemas dessas e de outras implementações do protocolo Dynamic Clamp foi feita por Prinz e colaboradores (Prinz, Abbott & Marder, 2004) e é uma referência obrigatória quando se começa a escolher qual a implementação mais adequada aos seus objetivos.

Embora a maioria das limitações de protocolos de Dynamic Clamp sejam de ordem técnica, como a velocidade limitada de computação e de aquisição de dados para a implementação de modelos ou a certeza da constância do período entre atualizações da corrente sináptica ou de uma condutância, uma limitação importante persistente e de

origem eletrofisiológica: o potencial de membrana é geralmente medido no soma do neurônio e a corrente é injetada, na melhor das hipóteses por um segundo eletrodo, também introduzido no soma. Entretanto, a maioria dos neurônios tem uma estrutura espacial bastante complicada e se comportam de modo muito diferente de um compartimento único isotônico.

7. Implementação de um protocolo Dynamic Clamp em sistema Linux em tempo real no Laboratório de Fenômenos Não-Lineares do IFUSP

Em nosso laboratório, o objetivo é utilizar o protocolo Dynamic Clamp para produzir circuitos híbridos (conectando neurônios artificiais e biológicos) e para interagir modelos computacionais com tecido vivo utilizando placas ADC/DAC comerciais. Além disso, como já tínhamos experiência anterior com a implementação em Windows (Pinto et al, 2001) queríamos rodar o ciclo de Dynamic Clamp em tempo real e não em tempo real como ocorre no Windows. Optamos por utilizar como base a versão RTLDC (Dorval et al., 2001) e alterá-la conforme necessário para implementar nossos modelos específicos de neurônios artificiais e também para controlar um circuito demultiplex analógico que permitisse conectar até 8 neurônios simultâneamente de modo similar ao empregado para conectar até 4 neurônios no Dynclamp4 para Windows.

7.1 - Instalando o RTLinux, o COMEDI e o RTLDC

Utilizamos inicialmente um computador Pentium 3, 500 MHz com memória de 256 MB. Para fazermos a implementação do Real Time Linux Dynamic Clamp (RTLDC) é necessário instalar no microcomputador o sistema operacional Linux. Após a instalação do sistema operacional, alguns pacotes devem ser instalados na seguinte ordem (todos podem ser carregados e instalados gratuitamente):

- RTAI ou RTLinux
- Comedi
- RTLDC

No início de nosso trabalho escolhemos uma das distribuições Linux, entre as disponíveis na época utilizamos o Fedora Core 1 (http://www.redhat.com). Esta distribuição do Linux utilizava o Kernel Linux versão 2.4.22-1.2115.nptl e o compilador gcc versão 3.2.

O próximo passo é instalar os pacotes de gerenciamento de tarefas em tempo real. O RTLDC tem a possibilidade de funcionar com duas versões de pacotes de Real Time: o RTAI ou o RTLinux. Como o RTAI possuía atualizações mais recentes e era mais

estável acabamos utilizando a versão RTAI rtai-24.1.10.tgz (http://www.rtai.org/). Infelizmente, para instalar a versão escolhida do RTAI era necessário usar o compilador gcc versão 2.95 mas o compilador disponível na versão do Fedora Core 1 instalado era gcc versão 3.2. Após diversas tentativas mal sucedidas de editar os arquivos fonte do RTAI para que compilassem com a versão disponível do gcc, acabamos instalando o gcc2.95 para resolver o problema.

Para estabelecer a comunicação e o controle da placa de aquisição de dados PCI-MIO-16E-4 (National Instruments) pelo RTLDC foi instalado o pacote Comedi (http://www.comedi.org). Este pacote encontra-se dividido em duas partes: Comedi e Comedilib e fornece os drivers necessários para o Linux acessar a maioria das placas de aquisição de dados de várias marcas comerciais, como National Instruments, Advantech, Analog Devices entre outros. Instalamos o comedi-0.7.68.tar.gz e o comedilib-0.7.18.tgz, e configuramos estes pacotes para o nosso modelo de placa.

Finalmente, instalamos e configuramos a versão 2.2.5 do RTLDC (rtldc-2.2.5.tar - http://www.bu.edu/ndl/rtldc.html) que não apresentou nenhum problema.

Após algum tempo desenvolvendo nossa implementação do RTLDC resolvemos substituir o computador por um AMD Athlon 2600 2.08GHz, 512Mb de memória, e tentamos utilizar versões mais atualizadas dos diversos pacotes. Instalamos a distribuição Fedora Core 3 com kernel Linux versão 2.6.9-1.667 e compilador gcc versão 3.4. Após diversas tentativas de fazer o sistema voltar a funcionar, descobrimos que o RTLDC era incompatível com a versão 2.6 do kernel. Voltamos a instalar o Fedora Core 1 mas encontramos problemas quando instalamos o RTAI. Após vários testes constatamos que o RTAI era incompatível com o processador AMD ou com a placa mãe que estavamos usando. A alternativa foi instalar o RTLinux que funcionou perfeitamente.

Quando iniciamos o trabalho não havia nenhum tipo de manual de instalação ou do usuário disponível para o RTLDC e tivemos que resolver vários problemas entrando em contado diretamente com o Prof. John White (um dos responsáveis pelo desenvolvimento do RTLDC) do Neuronal Dynamics Laboratory da Boston University. Apenas no meio deste trabalho um manual de instalação do RTLDC foi disponibilizado pelos autores (http://www.bu.edu/ndl/rtldc/doc/rtldc-install-manual.html). Entretanto, até este momento ainda não havia nenhum manual de usuário que explicasse como implementar novos modelos no RTLDC. Elaboramos um manual bastante simplificado para esta finalidade que se encontra na seção 7.4.

7.2 - Obtendo 8 sinais de corrente a partir de duas saídas analógicas de um DAC: multiplex analógico via software

A placa de aquisição de dados (PCI-MIO-16E-4, National Instruments), como a maioria das placas comerciais, apresenta um grande número de canais de entrada analógica (16 em nosso caso – IN0 a IN15) e apenas 2 canais de saída analógica (CH0 e CH1). Para possibilitar a conexão de ate 8 neurônios simultâneamente resolvemos multiplexar os sinais de corrente, amostrando-os em diferentes instantes nos dois canais de saída analógica.

Nossa implementação do Dynamic Clamp lê os potenciais de membrana dos 8 neurônios através dos canais de entrada analógica IN0 a IN7 da placa de aquisição de dados e calcula as 8 correntes (I0 até I7), de acordo com os modelos selecionados. Para o canal de saída analógico CH0 são enviados os sinais de correntes de canais pares: I0, I2, I4 e I6 e para o canal de saída analógico CH1 enviamos os sinais de correntes para os neurônios ímpares: I1, I3, I5 e I7.

A partir dos sinais compostos CH0 e CH1 e de um sinal digital de controle do tipo gatilho (*trigger*) para cada um dos 8 canais originais (D0 a D7), mostrados na Figura 22, usamos um circuito eletrônico (descrito na próxima seção) capaz de separá-los e produzir os 8 sinais diferentes de corrente para os neurônios.

Para produzir as 8 correntes de saída e os sinais de controle (D0 a D7) para o circuito demultiplex as seguintes modificações foram feitas no pacote RTLDC original (neste texto todos os caminhos de diretórios e arquivos se referem ao diretório onde foi instalado o RTLDC):

- No arquivo include/daq_spec.h: o parâmetro número máximo de canais de saída MAX_AO_CHANNELS foi modificado de 2 para 8 canais. Também adicionamos um novo parâmetro para indicar o número real de canais de saída, REAL_MAX_AO_CHANNELS igual a 2. Uma listagem do arquivo daq_spec.h completo com todas as alterações feitas anotadas encontra-se no Apêndice 1.

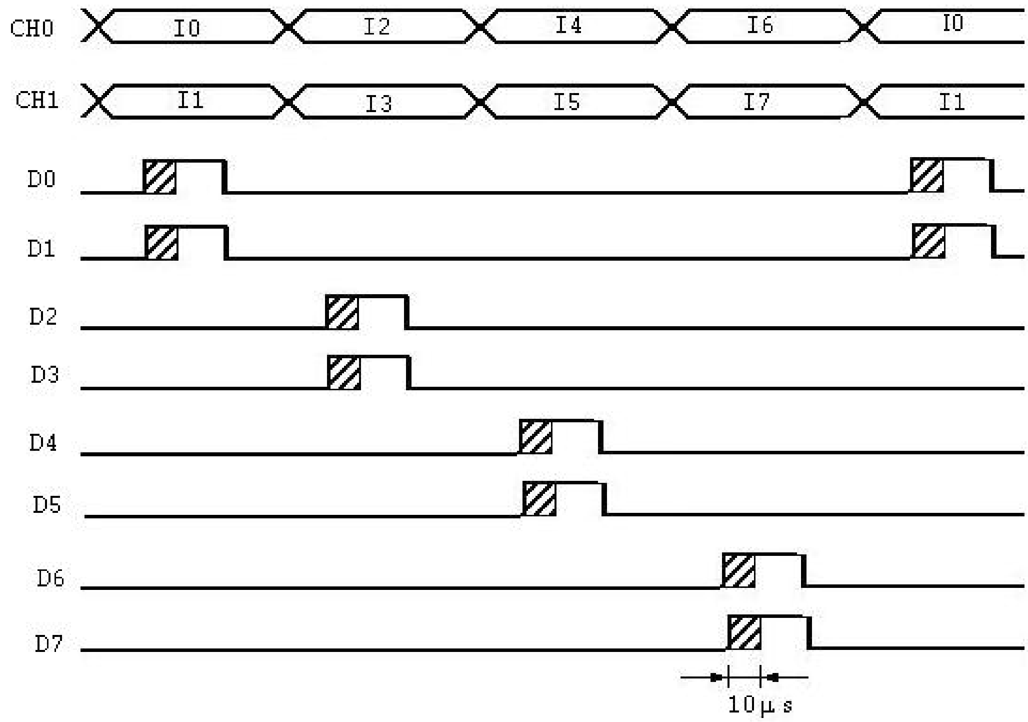


Figura 23: Diagrama temporal dos sinais analógicos e de controle do demultiplex. As saídas analógicas da placa DAC, CH0 (CH1), são usadas para amostrar as voltagens de saída correspondentes às correntes I0, I2, I4 e I6 (I1, I3, I5 e I7) com relação aos sinais de controle digitais, pulsos de controle (D0-D7). Como o programa é síncrono a freqüência de atualização teve que ser limitada a 3KHz. Os sinais de *sample* para cada um dos 8 canais de saída são produzidos por monoestáveis (U3 - U10) durante o tempo marcado pelas regiões haxuradas.

- <u>No programa principal</u> realtime/rtldc.c: alteramos a forma de escrita do sinal nas saídas analógicas e incluimos novas linhas para produzir saídas digitais sincronizadas com o sinal analógico. Tivemos que alterar várias definições do programa e modificar várias funções, como as de escrita e leitura. A interface gráfica também foi modificada para facilitar ao usuário a mudança de valores de parâmetros. A escrita dos sinais analógicos na saída foi feita de forma a sempre enviar um par simultâneo de correntes às saídas analógicas CH0 e CH1: (I0 e I1; I2 e I3; I4 e I5; I6 e I7) e um par de sinais digitais de comando (D0 e D1; D2 e D3; D4 e D5; D6 e D7), conforme mostrado na Figura 22.

Uma listagem do programa rtldc.c que passou a se chamar rtldc_demultiplex.c com todas as alterações feitas anotadas encontra-se no Apêndice 2.

7.3 - Demultiplex via hardware

Projetamos e construímos um circuito eletrônico para demultiplexar os sinais analógicos CH0 e CH1 e gerar as 8 saídas de corrente. Nosso demultiplex é baseado em um circuito integrado SMP04 (Analog Devices) que opera de modo conhecido como *sample & hold*. Na Figura 23 é mostrado o esquema elétrico do demultiplex analógico de um canal multiplexado (CH0 ou CH1) para 4 saídas. Para produzir os 8 sinais de corrente utilizamos 2 circuitos demultiplex idênticos, um para cada sinal multiplexado.

O circuito integrado SMP04 de sample & hold é composto basicamente por capacitores internos que são carregados com a tensão de entrada quando o sinal de comando *sample* é ativado (nível lógico 1 TTL) e mantém a tensão de saída igual à tensão armazenada no capacitor durante a duração do comando *hold* (nível lógico 0 TTL).

O sinal multiplexado CH0 (ou CH1) passa inicialmente por um amplificador operacional com ganho –1/2 para que sua amplitude máxima (±10V) seja reduzida para ±5V, dentro dos níveis de trabalho do SMP04 (±6V) e é ligado em todas as entradas analógicas do sample & hold. Os sinais de comando pares D0-D6 (ou os ímpares D1-D7) acionam multivibradores astáveis TTL do tipo 74LS123. Estes circuitos foram preparados de modo a produzir um pulso nível 1 TTL com duração de 10 μs que aciona o comando *sample* da respectiva porta do SMP04 (este tempo é o mínimo necessário para garantir que a entrada seja amostrada pelo sample & hold). Na ausência de pulsos nesta entrada o canal do SMP04 fica em modo hold. Como cada sinal de comando é ligado a uma diferente porta do sample & hold, o sinal presente na entrada passa apenas pela porta que teve seu *sample* ativado e conseguimos separar as 4 componentes distintas possíveis. Após a separação em componentes as saídas do SMP04 são ligadas a amplificadores com ganho –2 para restaurar a amplitude original do sinal. Estes amplificadores de saída possuem 2 potenciômetros multivoltas onde podemos fazer um ajuste fino do ganho (para garantir que o circuito não altere a amplitude do sinal de entrada) e um ajuste fino do off-set onde cancelamos qualquer nível DC introduzido no processo. Em todos os amplificadores utilizamos o operacional LM308, bastante comum em instrumentação, com baixo custo, boa estabilidade e precisão.

Inicialmente desenvolvemos um protótipo do circuito em protoboard para testes e ajustes dos valores dos componentes. O circuito final total, composto por duas placas idênticas, foi montado em placas de circuito impresso dupla face em fibra de vidro produzidas em uma frezadora comandada por computador. O layout das placas foi

projetado e produzido com o auxílio da versão gratuita, limitada a pequenos projetos, do software EAGLE (Cadsoft Computer, Inc.).

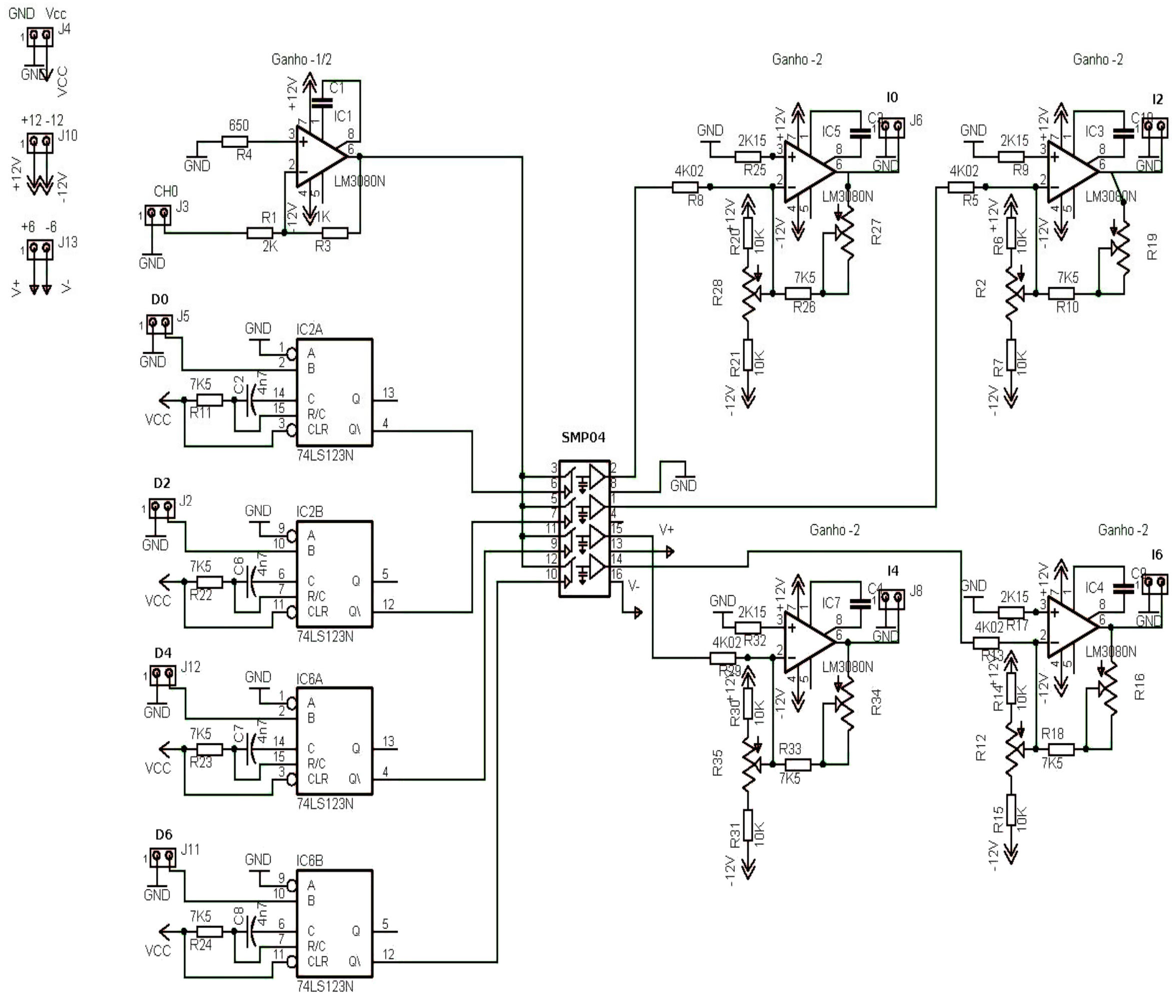


Figura 24: Esquema elétrico do circuito do demultiplex analógico. CH0 é o canal de entrada do sinal analógico multiplexado. A demultiplexação é feita por amostragem em um circuito tipo sample&hold (SMP04) controlado pelos sinais digitais D0 – D6. Como o sample&hold trabalha com amplitude de sinal de entrada menor que a amplitude do sinal multiplexado CH0, o sinal passa inicialmente por um amplificador com ganho -1/2 e as saídas do sample&hold passam por um amplificador de saída com ganho -2 para reconstruir o sinal de corrente I0 – I6 com a amplitude original. O circuito demultiplex dos canais de corrente ímpares a partir de CH1 é idêntico.

A montagem dos circuitos foi feita com componentes de precisão (resistores 1% quando possível) para minimizar efeitos de variação da temperatura de operação do circuito. Após a montagem, mostrada na Figura 24, realizamos vários testes para verificar a presença de ruído: com a proximidade entre sinais digitais rápidos de comando e sinais analógicos de baixa amplitude no integrado de sample & hold, uma das preocupações é

que os sinais digitais induzam pequenos pulsos que podem contaminar os sinais de corrente. Diversos cuidados foram tomados no projeto do layout para minimizar estes efeitos, como a colocação de malhas de terra ao redor das trilhas condutoras de sinais analógicos (sempre que possível) e procurar distanciar o máximo possível as trilhas de sinais analógicos das trilhas digitais. Nenhum ruído significante foi observado nos diversos testes realizados.

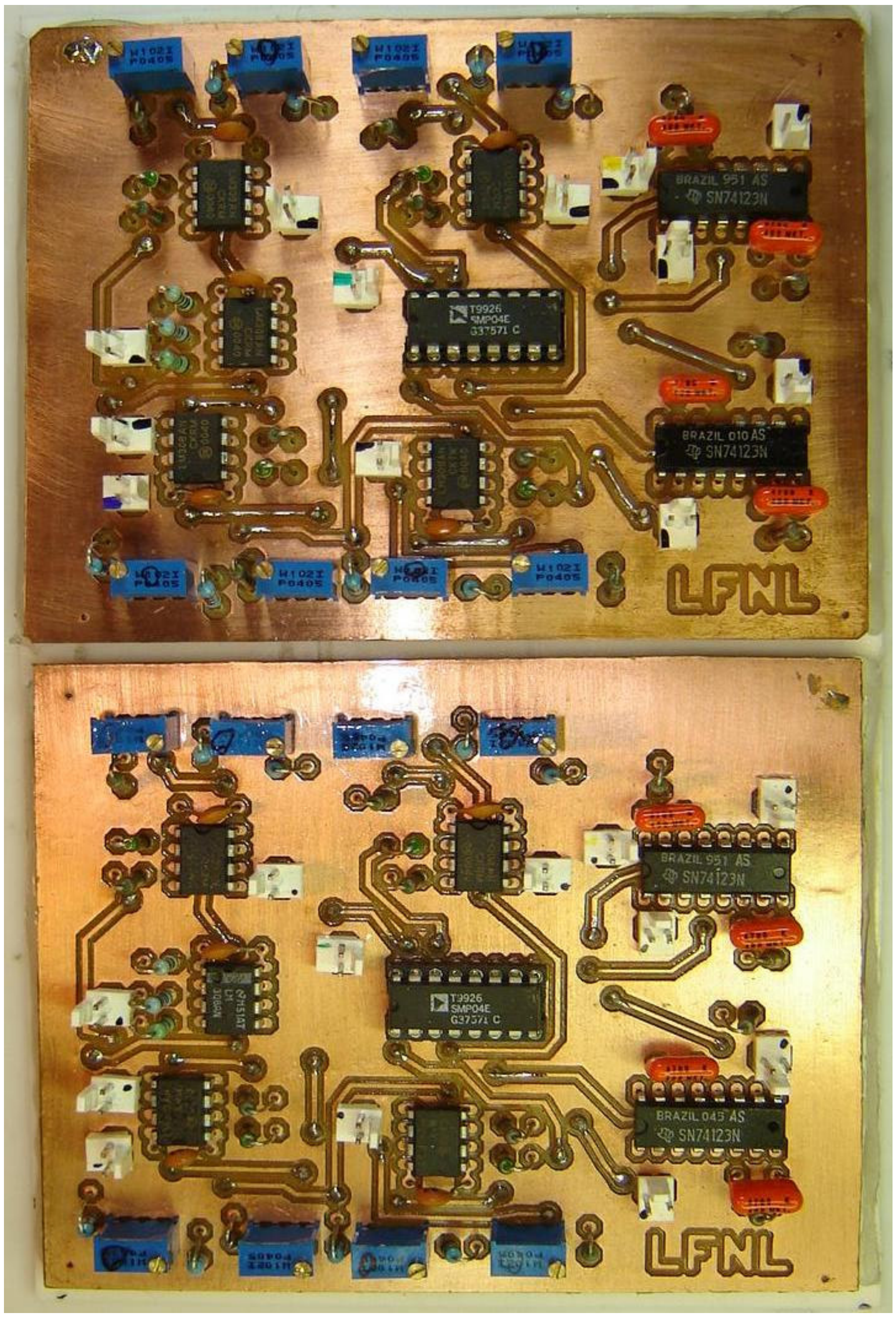


Figura 25: Imagem da placa de circuito impresso do demultiplexador no seu estado final para testes. Usamos duas placas idênticas, uma para demultiplexar as correntes pares (I0, I2, I4, I6) e outra para demultiplexar as correntes ímpares (I1, I3, I5, I7).

Para exemplificar o funcionamento do circuito, aplicamos os sinais gerados por dois neurônios eletrônicos tipo HR (Capítulo 4) às entradas analógicas IN0 e IN2 da placa de aquisição de dados e programamos o RTLDC para copiar estas entradas nas saídas de corrente I0 e I2. Apesar de produzirmos apenas 2 correntes, todos os 8 canais de saída estavam habilitados, e por isso uma taxa de atualização máxima de 3 kHz foi utilizada (para 4 canais podemos usar 6-7 kHz de taxa de atualização). Na Figura 25 temos nos dois traços superiores o sinal dos neurônios eletrônicos; no traço intermediário temos a saída multiplexada CH0 e nos traços inferiores as saídas demultiplexadas I0 e I2.

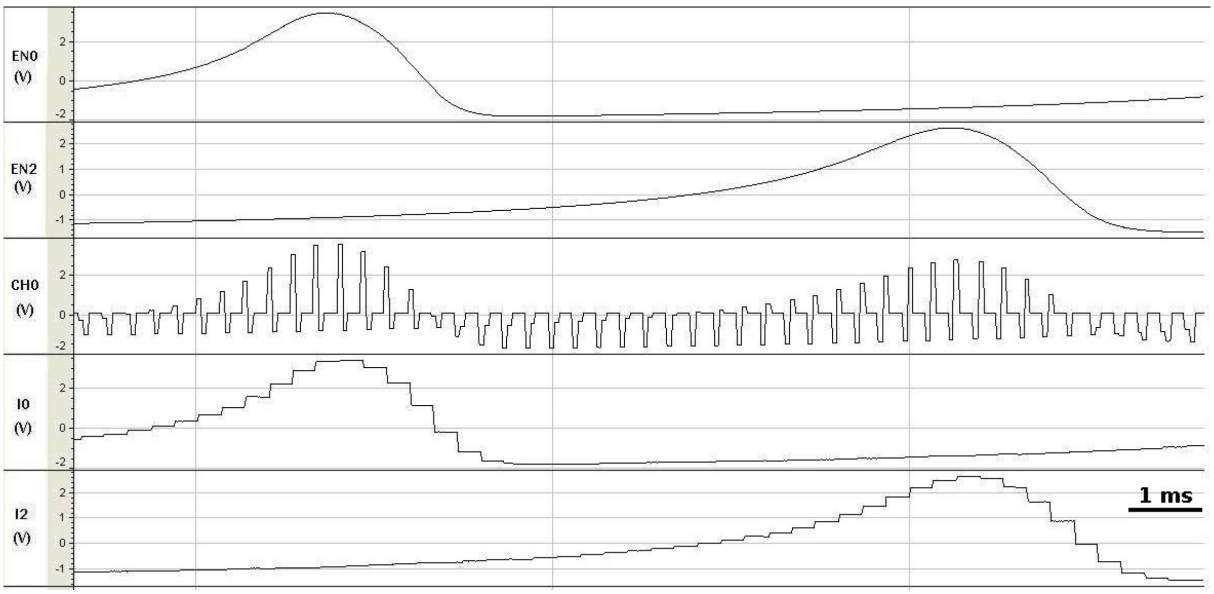


Figura 26: Exemplo de funcionamento do circuito demultiplex. Os sinais de dois neurônios eletrônicos (EN0 e EN2) são colocados nas entradas analógicas da placa de aquisição. Em CH0 temos a saída multiplexada da placa PCI-MIO-16E-4. Nas saídas I0 e I2 do circuito demultiplex temos os sinais após sua separação.

Podemos observar que os sinais de EN0 e EN2 estão misturados em CH0. Para separá-los, o RTLDC envia os sinais de controle D0 e D2 para o circuito de demultiplexação. Ao receber o sinal D0 um comando *sample* é produzido e faz com que o circuito sample & hold que gera I0 armazene o valor de CH0 (que contém neste instante IN0). Em seguida, o RTLDC coloca o valor da entrada IN2 (EN2) em CH0 e o pulso de comando D2 é produzido, fazendo com que o sample & hold que gera I2 armazene o valor de IN2. Este ciclo é repetido à taxa de 3 kHz como descrito anteriormente.

Na Figura 26 mostramos em detalhe vários ciclos de funcionamento do circuito demultiplex. Por simplicidade mostramos apenas a entrada IN0 (EN0), o sinal multiplexado CH0, o sinal de comando D0 e a saída de corrente I0. A cada ativação do pulso digital D0, o sinal analógico na saída I0 é atualizado com o valor que estiver em

CH0, mantendo-se neste valor até que outro pulso de comando D0 seja recebido. Assim, o sinal contínuo de entrada (EN0) é transformado em um sinal discreto com pequenos degraus mas ainda bastante semelhante à entrada. Como este sinal de corrente é introduzido no corpo celular dos neurônios e este funciona como um integrador e filtro passa-baixas, na maioria das aplicações os neurônios acabam entendendo o sinal como sendo suave, apesar dos pequenos degraus.

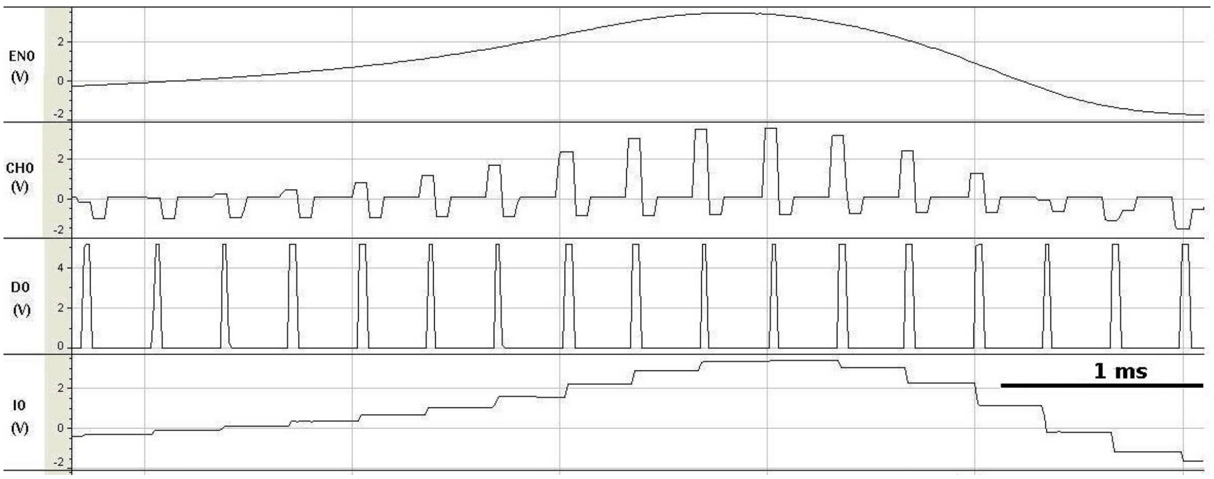


Figura 27: Em uma escala mais ampliada da Figura anterior, a cada ativação do pulso digital D0 o sinal analógico presente em CH0 (que contem os sinais de EN0 e EN2 misturados) é atualizado na saída I0.

7.4 - Implementando novos modelos no RTLDC

Para implementar novos modelos no RTLDC três arquivos devem ser criados: (neste exemplo vamos supor que o nome do modelo seja ***<seu_modelo>***)

<seu_modelo>.c - que deverá conter o código fonte do modelo matemático (escrito em linguagem de programação C) utilizado para calcular as correntes de saída. Os parâmetros do modelo, nomeados no arquivo *<seu_modelo>.c*, devem ser declarados no inicio deste arquivo e normalmente são declarados de forma static para não sobrepor variáveis de outros arquivos. *<seu_modelo>.c* deverá possuir obrigatóriamente 4 rotinas:

- *init_module()* - é uma rotina exigida pelo núcleo do RTLDC para inicializar o modelo quando ele é carregado. Esta rotina faz a alocação de memória para guardar a informação do modelo e notifica ao RTLDC que o modelo foi lido com sucesso. Basta copiá-la de um modelo que já

está funcionando e alterar as linhas:

```
shmem->main_fxn     = & seu_modelo_main;
shmem->update_fxn   = & seu_modelo_update;
```

para que elas apontem para os nomes das rotinas main e update dentro do arquivo <seu_modelo>.c.

- *cleanup_module()* - faz o oposto da rotina anterior, liberando a memória alocada, encerrando o uso do modelo pelo RTLDC. Novamente é suficiente copiá-la de um modelo que já esteja funcionando uma vez que como a *init_module* não requer nenhuma alteração na maioria dos casos.

- *seu_modelo_main()* – rotina onde é definido o modelo matemático a ser calculado, podendo ser desde uma simples soma até um cálculo integral. O RTLDC envia como parâmetros os valores lidos das tensões dos neurônios. Através do parâmetro "k" o RTLDC indica qual o rótulo dado a este modelo específico, já que é permitido que vários modelos sejam carregados e operem simultaneamente;

- *seu_modelo_update()* – esta rotina inicializa os parâmetros do modelo logo depois que este foi inicializado por *Init_module()*. Esta rotina também é chamada quando o usuário modifica os valores dos parâmetros na interface gráfica do RTLDC. Novamente o parâmetro "k" é usado para atualizar apenas os parâmetros do modelo específico.

<seu_modelo>.model - contém a ligação entre o nome dos parâmetros mostrados na interface gráfica do programa às variáveis definidas no modelo, indicando seu número e o nome pela ordem de apresentação.

Makefile - este arquivo contém as regras de compilação do modelo e os diretórios onde estão os arquivos .h necessários, onde devem ser colocados os arquivos executáveis, os objetos, etc... Este arquivo Makefile pode ser facilmente adaptado a partir do Makefile de outro modelo já existente e normalmente só é necessário alterar a primeira linha fazendo:

MODULE = <seu_modelo>

Para compilar e executar um modelo, o arquivo ***<seu_modelo>.c*** deve ser colocado no diretório ***src/model/<seu_modelo>/*** onde também deve ficar o arquivo ***Makefile***, usado pelo comando "make" para compilar e copiar os arquivos nos diretórios correspondentes. Os comandos usados do make são: "*make*" para compilar o modelo e "*make clean*" para apagar arquivos temporários.

O arquivo ***<seu_modelo>.model*** com as informações para a interface gráfica deve ser colocado em um diretório chamado ***models/<seu_modelo>/***. Os diretórios acima devem ser criados a partir do diretório onde se encontra instalado o RTLDC.

A seguir colocamos o exemplo de um modelo chamado **teste_copia** criado para copiar o sinal da entrada analógica 0 na saída analógica 0 com a possibilidade de fazer um ajuste de ganho (inicialmente = 1) e offset (inicialmente = 0).

Arquivo **teste_copia.c** :

```
// ---------------início do arquivo teste_copia.c  --------------------------------------------------

#include <model.h>   // inclusão das bibliotecas com diversas funções utilizadas //

// declaração de parâmetros //
static int    ai[MAX_FUNCTIONS];
static int    ao[MAX_FUNCTIONS];
static double gain[MAX_FUNCTIONS];
static double offset[MAX_FUNCTIONS];

// ------  rotina main -------------------------------------------------------------------//
extern void teste_copia_main (double *data,int mem_index)
{
  static int k;
  k = mem_index;

  // a seguir é feita a escrita do sinal de entrada input(ai[k])  //
  // multiplicado por um ganho gain[k] e por um ajuste de offset = offset[k] //
  // observe que para não interferir com outros modelos que podem estar //
  // rodando e usando a mesma saída analógica, a maneira correta de gerar //
  // uma saída é: output(ao[k]) = output (ao[k]) + função realizada pelo modelo //
```

```
output(ao[k]) = output(ao[k])+gain[k]*input(ai[k])+offset[k];
}

// ------  rotina update ------------------------------------------------------------------//
extern void teste_copia_update(double *data,int mem_index,int first_time)
{
  static int k;
  k = mem_index;

  if(first_time)     // utiliza valores default quando o usuário carrega o modelo //
      {
    shmem->parameter[k][0] = 0;   // canal de entrada = 0
    shmem->parameter[k][1] = 0;   // canal de saída = 0
    shmem->parameter[k][2] = 1;   // ganho = 1
    shmem->parameter[k][3] = 0;   // offset = 0
  }
  ai[k] = (int)shmem->parameter[k][0]; // canal de entrada analógica
  ao[k] = (int)shmem->parameter[k][1]; // canal da saída analógica
  gain[k]    = shmem->parameter[k][2];
  offset[k]  = shmem->parameter[k][3];
}

// ------  rotina init_module ------------------------------------------------------------------//
int init_module(void)
{

#if defined RTLINUX
  shmem = (rtldc_shm *)mbuff_alloc(SHM_ADDR,sizeof(rtldc_shm));
#elif defined RTAI
  shmem = (rtldc_shm *)rtai_kmalloc(nam2num(SHM_ADDR),sizeof(rtldc_shm));
#else
#error "Must #define RTLINUX or RTAI"
#endif

      // a seguir são indicados para o RTLDC os  //

      // nomes das rotinas main e update no modelo //

  shmem->main_fxn    = & teste_copia_main;
  shmem->update_fxn  = & teste_copia_update;

  return 0;
}


// ------  rotina cleanup_module ------------------------------------------------------------------//
void cleanup_module(void)
{

#if defined RTLINUX
  mbuff_free (SHM_ADDR,(void *)shmem);
#elif defined RTAI
  rtai_kfree(nam2num(SHM_ADDR));
#else
#error "Must #define RTLINUX or RTAI"
#endif
}
// --------Fim do arquivo  teste_copia.c-------------------------------------------------------
```

Arquivo **teste_copia.model** :

```
ModelName= teste_copia
NumParams=4   // número de parâmetros utilizados //
Param[0]=Input Chan
Param[1]=Output Chan
Param[2]=Gain
Param[3]=Offset
NumStates=0
// com base neste arquivo o RTLDC irá criar uma interface //
// gráfica com campos onde poderão ser alterados os valores //
// dos parâmetros: Input Chan, Output Chan, Gain e Offset //
```

Arquivo **Makefile** :

```
MODULE = teste_copia

#### Flags ####

include ../../../.config
include ../../../.cflags

CC      = gcc
CFLAGS  = $(RTL_CFLAGS)
HEADERS = ../../include/rtldc.h ../../include/daq_spec.h
INCLUDE = -I../../include $(RTL_INC) $(COMEDI_INC)

.PHONY = all clean


#### Build Rules ####

all: ../../../modules/$(MODULE).o

clean:
	rm -f *~
	rm -f \#*
	rm -f *.o
	rm -f ../../../modules/$(MODULE).o

../../../modules/$(MODULE).o: $(MODULE).c $(HEADERS)
	$(CC) $(CFLAGS) $(INCLUDE) -c $(MODULE).c -o
../../../modules/$(MODULE).o
```

---

8. Experimentos e Resultados

Realizamos diversos experimentos para testar o nosso equipamento de Dynamic Clamp. Nestes experimentos utilizamos modelos de sinapses químicas como o descrito no Capítulo 6, neurônios eletronicos (ENs) tipo HR como os descritos no Capítulo 4 (ajustados para apresentar comportamento caótico quando isolados) e o modelo de neurônio estocástico descrito no capítulo 5 que foi implementado na forma de modelo do RTLDC pelo estudante Pedro Carelli (Carelli et al, 2005). Um segundo computador que possui um sistema de aquisição de dados para eletrofisiologia Digidata 1322 e software Axoscope 9 (ambos da Axon Instruments) foi usado para digitalizar os diversos sinais a uma taxa de 10 kHz e armazená-los em disco para futura análise.

Para explorar a capacidade de conectar mais de dois neurônios do nosso RTLDC, nos concentramos em experimentos formando pequenas redes artificiais com 3 ou 4 neurônios onde pudemos observar diversas características dos mecanismos de oscilação que são encontrados em CPGs biológicos (Muloney & Selverston, 1974; Selverston & Moulins, 1986; Sharp, Skinner & Marder, 1996).

As diversas sinapses entre neurônios eletrônicos foram implementadas com diferentes valores de condutância *G* (informados explicitamente nos experimentos) e com os demais parâmetros fixados (a não ser que esteja explicitamente citado no texto) nos seguintes valores, baseados nas sinapses encontradas nos CPGs de crustáceos (Selverston et al., 2000): potencial reverso $V_{syn} = -3{,}5V$ (um pouco abaixo do valor mínimo do potencial de membrana dos ENs para garantir a inibição), $V_{slope} = 1{,}0V$ (para saturar levemente a corrente durante o pico dos potenciais de ação do neurônio pré-sináptico), potencial de gatilho $V_{thres} = -1{,}5V$ (um pouco abaixo do limiar dos potenciais de ação para simular sinapses do tipo *"graded"* que liberam um pouco de neurotransmissor quando o neurônio pré-sináptico despolariza mesmo sem a presença de um potencial de ação), constante de tempo sináptica $\tau_s = 10\ ms$.

Os sinais de corrente sináptica mostrados nos diversos gráficos são ligados nos ENs em uma entrada que tem uma relação 1:1 com a corrente adimensional I_{syn} que entra na equação diferencial para *x* (Pinto et al., 2000). Isto significa que se em um determinado instante o valor do sinal de corrente mostrado no gráfico é igual a $-2{,}7V$, então $I_{syn} = -2{,}7$.

8.1 – CPGs artificiais com 3 neurônios

8.1.1 – Inibição em anel unidirecional

Os neurônios eletrônicos EN0, EN1 e EN2 foram conectados por três sinapses químicas inibidoras implementadas no RTLDC de forma a produzir uma topologia em anel, como mostrado na Figura 27. Cada sinapse química inibidora é representada por uma curva que sai do neurônio pré-sináptico e chega ao neurônio pós sináptico onde há um pequeno círculo. Nesta topologia EN0, ao disparar, inibe EN1 que, ao disparar, inibe EN2 que, ao disparar, inibe EN0.

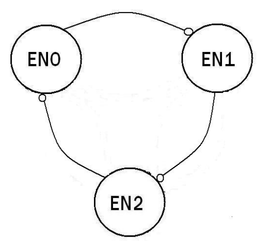


Figura 28: Anel formado por três neurônios conectados com sinapses químicas inibidoras e idênticas.

Na Figura 28 mostramos o resultado obtido neste experimento quando, a partir de uma condição inicial em que o RTLDC está desligado, as sinapses são todas ligadas com o mesmo valor de condutância *G = 4nS*. Na Figura 28 e em outros gráficos *EN0, EN1 , ... ENn* representam o "potencial de membrana" – valor da variável *x* - dos ENs, enquanto *I0, I1, ... e In* representam o valor das correntes I_{syn} injetadas nos respectivos ENs. Enquanto as sinapses estão desligadas (*0 < t < 6.8s*) os ENs se comportam de modo caótico e independente. Em *t = 6.8s* as sinapses do RTLDC são ativadas e após um curto transiente (menor que *1s*) os neurônios passam a exibir um rítmo trifásico e periódico de disparo, em que os neurônios se sucedem sempre na mesma fase (sequência EN0-EN1-EN2-EN0-...) apresentando bursts com 4 spikes/burst a uma frequência de ~ 6.5 bursts/s.

Um fato interessante na topologia de CPGs biológicos é que estes geralmente são formados por pares de neurônios conectados por mútua inibição (*half center oscillators*). Circuitos biológicos formados por pares de neurônios com mútua inibição são capazes de alterar bastante a frequência das oscilações da rede quando a condutância sináptica é alterada devido à neuromodulação (Selverston e Moulins, 1986). *Half-center oscillators* são considerados por muitos autores como os principais blocos constituintes de circuitos geradores de rítmos como os CPGs.

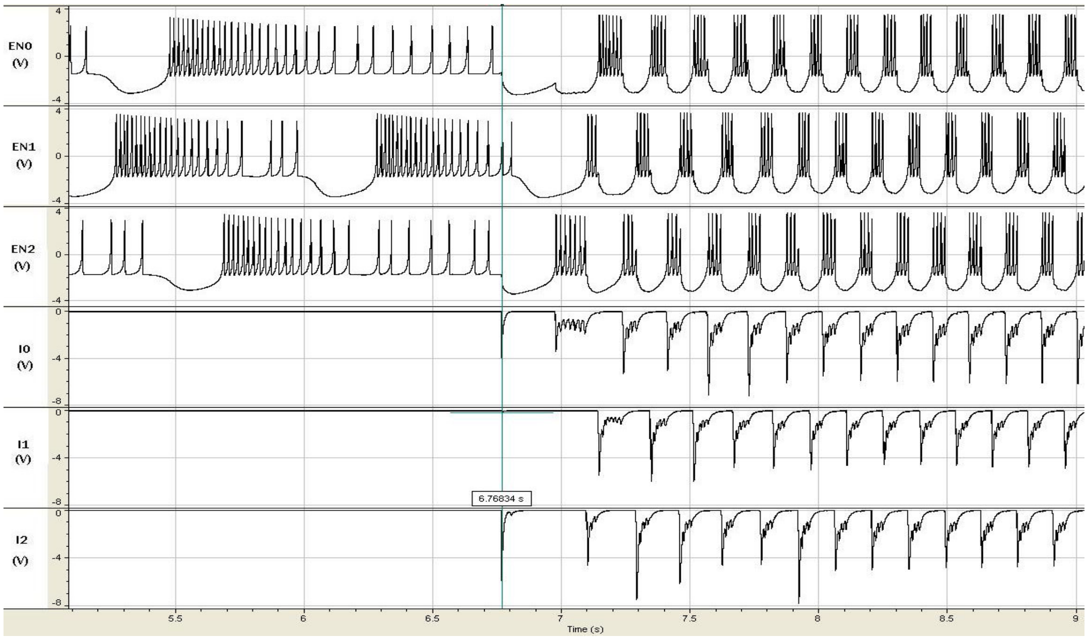


Figura 29: Potenciais de membrana dos neurônios eletrônicos e correntes sinápticas produzidas pelo RTLDC quando os neurônios são conectados em uma topologia em anel e todas as sinapses tem a mesma condutância *G = 4 nS.* Inicialmente o RTLDC está desativado e os neurônios se comportam de modo irregular e independente, após t ~ 6,7 s as sinapses são ativadas e um ritmo oscilatório periódico é rapidamente estabelecido.

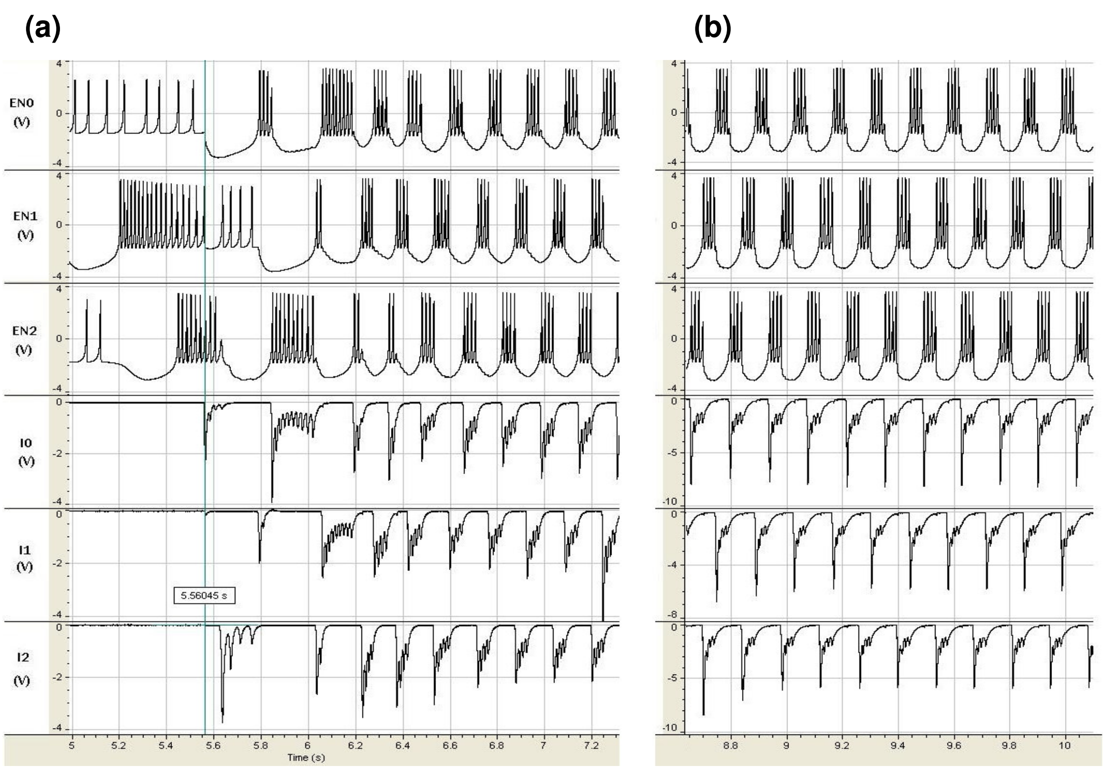


Figura 30: Potenciais de membrana dos neurônios eletrônicos e correntes sinápticas produzidas pelo RTLDC quando os neurônios são conectados em uma topologia em anel e todas as sinapses tem a mesma condutância **(a)** *G = 1.5 nS* e **(b)** *G = 6 nS*. Apesar da amplitude do pico negativo de corrente aumentar com a condutância, a frequência em bursts/s e o número de spikes/burts não são alterados. Em **(b)** não é mostrado o transiente inicial.

Para investigar o efeito de alterar a condutância em uma topologia em anel unidirecional (que não apresenta mútua inibição) mudamos o valor da condutância *G* das sinapses de nossa rede neural para *1.5 nS* e *6 nS* (resultados mostrados na Figura 29). Para todos os valores de *G* que utilizamos entre *1.5 nS* e *6 nS*, apesar de se observar que a amplitude do pico negativo da corrente cresce com *G,* nenhuma alteração na frequência em bursts/s ou no número de spikes/burst é observada. Ou seja, na topologia em anel sem mutua inibição estas características não dependem da intensidade da conectividade sináptica. Se um neuromodulador alterar os valores de *G* em um CPG com topologia do tipo anel unidirecional nenhuma mudança de comportamento ocorre no circuito.

8.1.2 – Anéis com mútua inibição

Iniciamos a investigação do efeito da presença de mútuas inibições em CPGs considerando o mesmo circuito em anel unidirecional mostrado na seção anterior com a inclusão de uma sinapse entre EN1 e EN0 conforme mostrado na Figura 30. Neste exemplo configuramos todas as sinapses com a mesma condutância *G = 4nS*.

O comportamento da rede em anel quando a sinapse de EN1 para EN0 é ativada é mostrado na Figura 31. Para *t < 13.6s* a sinapse EN1->EN0 está desligada, a rede tem a topologia de anel unidirecional, e seu comportamento é idêntico ao mostrado na seção anterior. Em *t ~ 13.6s* a sinapse EN1->EN0 é ligada e o comportamento da rede muda drasticamente. Apesar de a nova sinapse ter o mesmo valor de condutância das sinapses que já estavam no circuito a frequência de oscilação cai de ~6.7 bursts/s para ~1.8 bursts/s (quase ¼ da frequência original) e os neurônios que possuiam praticamente o mesmo ciclo de trabalho e apresentavam sinais muito parecidos apesar de defasados, passam a apresentar ciclos completamente distintos de atividade.

Acrescentando uma nova sinapse com condutância *G = 4nS*, que estabelece mútua inibição entre EN1 e EN2, como mostrado na Figura 32, obtivemos o comportamento observado na Figura 33. Inicialmente, todas as sinapses do RTLDC estão desativadas e os neurônios apresentam comportamento intrinsecamente caótico e independente. Em *t ~ 6,7s* as sinapses são ativadas e o circuito mostrado na Figura 32 é estabelecido.

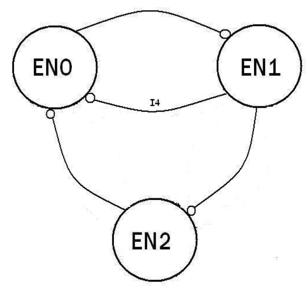


Figura 31: Três neurônios conectados com sinapses químicas inibidoras idênticas formando um anel unidirecional com uma sinapse extra entre EN1 e EN0 – mútua inibição.

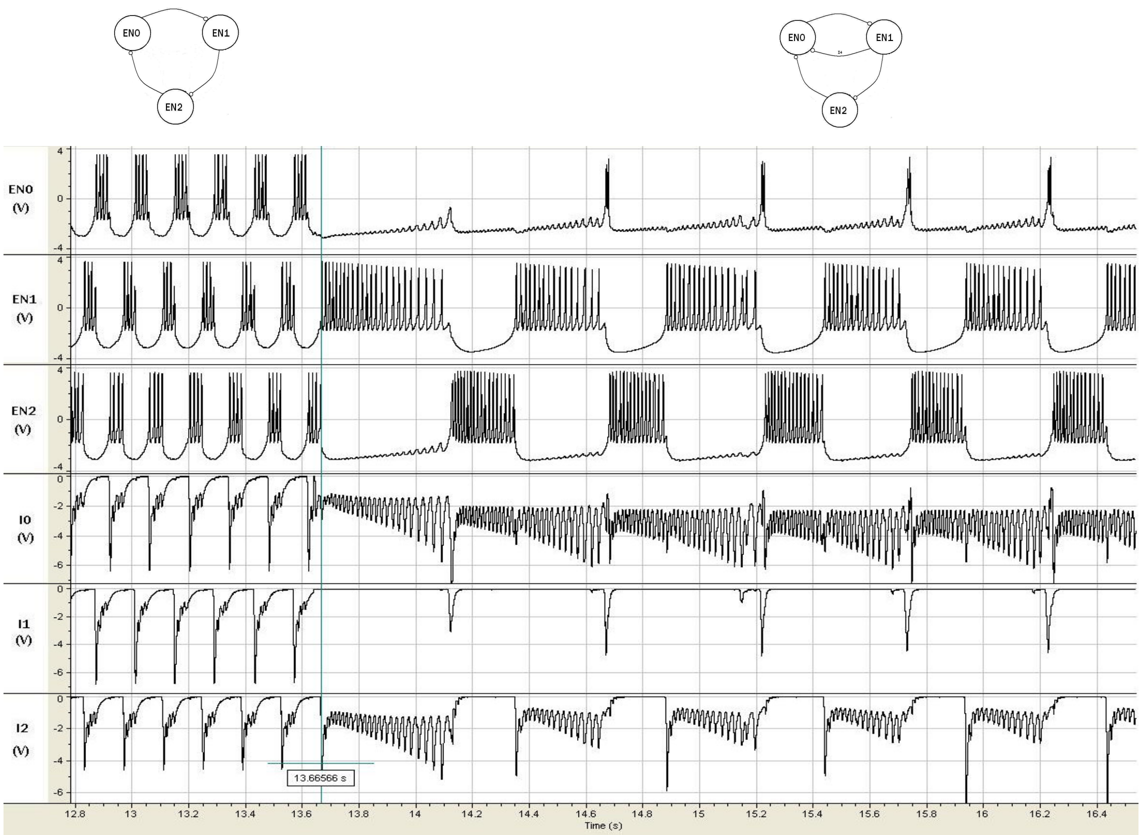


Figura 32: Potenciais de membrana dos neurônios eletrônicos e correntes sinápticas produzidas pelo RTLDC quando os neurônios são conectados em uma topologia em anel. No início o anel é simétrico e o comportamento é o mesmo mostrado na seção anterior. Em *t ~ 13.6s* uma sinapse extra entre EN0 e EN1, que passam a apresentar mútua inibição, é ativada e o comportamento se altera completamente: a frequência passa de ~6.7 bursts/s para ~1.8 bursts/s (quase ¼ da frequência original) e os neurônios que possuiam todos o mesmo duty cycle passam a apresentar ciclos distintos de atividade. Todas as sinapses tem *G = 4 nS*.

Assim que as sinapses são ligadas em *t ~ 6.7s* um ritmo de oscilação bastante assimétrico é rapidamente estabelecido. Comparando com a frequência do caso anterior (com apenas uma mútua inibição) a frequência diminui para 0.7 bursts/s. Apesar da nova sinapse estar ligada entre os neurônios EN2 e EN1 é em EN0 que se observa a maior mudança de comportamento, este passa a disparar duas vezes por ciclo, um disparo

quase normal que começa no final da atividade de EN2 e um disparo muito rápido (com apenas um spike) após o fim dos disparos de EN1.

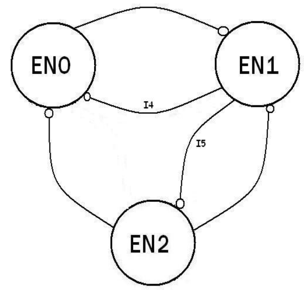


Figura 33: Três neurônios conectados em anel com duas sinapses que estabelecem mútua inibição entre os neurônios EN0 e EN1 e entre EN1 e EN2.

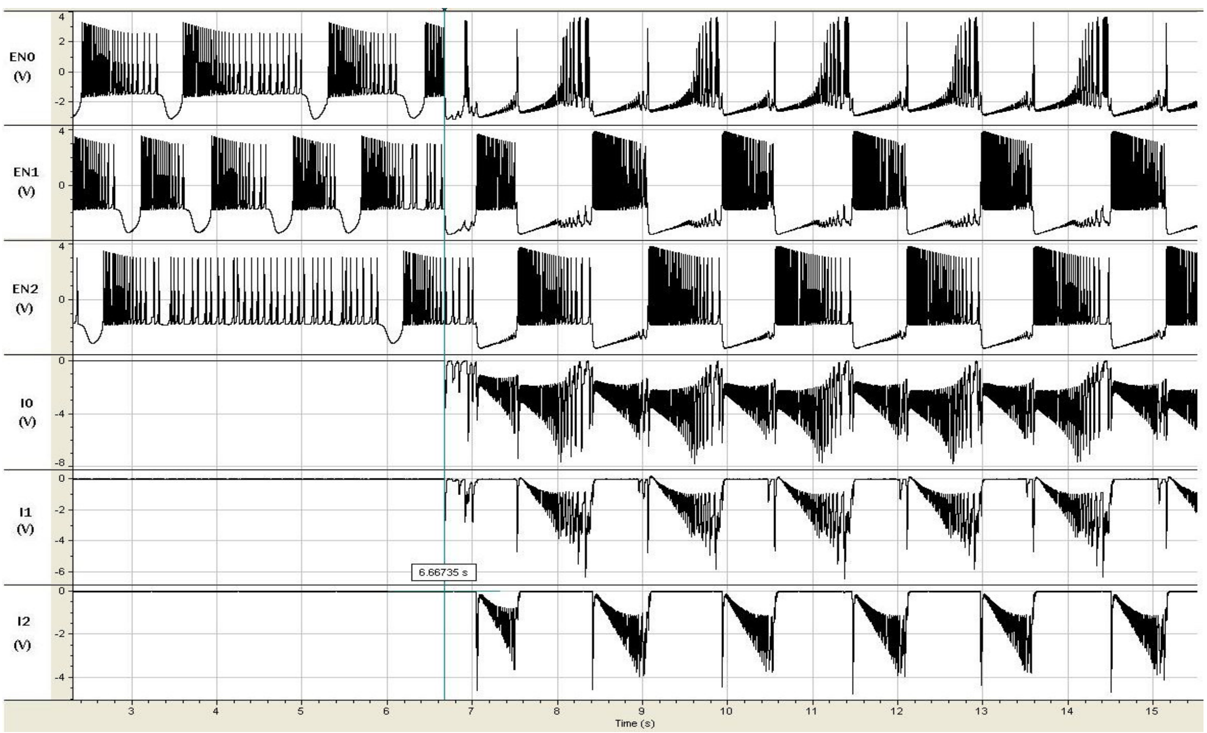


Figura 34: Potenciais de membrana dos neurônios eletrônicos e correntes sinápticas produzidas pelo RTLDC quando os neurônios são conectados em uma topologia em anel com duas sinapses extra que estabelecem mútua inibição entre EN0 e EN1 e entre EN1 e EN2. No início todas as sinapses estão desligadas e o comportamento dos neurônios é independente e caótico. Em $t \sim 6.7s$ as sinapses são ligadas e rapidamente um ritmo de oscilação bastante assimétrico é estabelecido. A frequência diminui (quando comparada com o caso da Figura 31) para 0.7 bursts/s. Todas as sinapses tem $G = 4\ nS$.

Finalmente, testamos uma última topologia com todos os neurônios conectados com mútua inibição. Como nos exemplos anteriores, as sinapses tem os mesmos parâmetros fixos e $G = 4\ nS$. O esquema de ligação é mostrado na Figura 34 e os sinais obtidos estão na Figura 35.

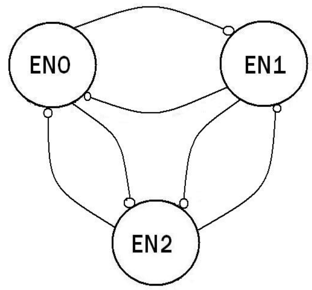


Figura 35: Três neurônios conectados com mútua inibição em uma topologia em anel. Todas as sinapses são idênticas (*G = 4nS*).

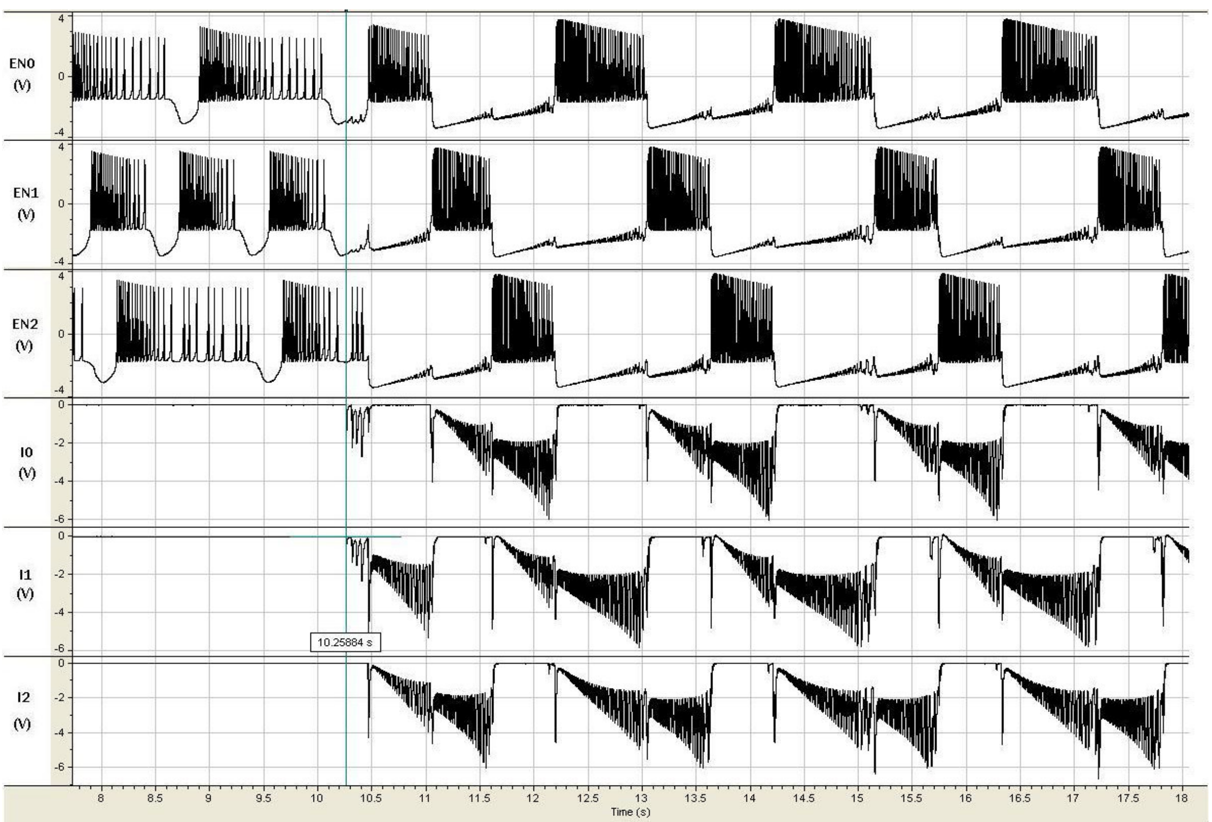


Figura 36: Potenciais de membrana dos neurônios eletrônicos e correntes sinápticas produzidas pelo RTLDC quando os neurônios são todos conectados com mútua inibição em uma topologia em anel. Inicialmente as sinapses estão desligadas e o comportamento dos neurônios é independente e caótico. Em *t ~ 10.2s* as sinapses são ligadas e um ritmo trifásico periódico é rapidamente estabelecido. A frequência de oscilação da rede é aproximadamente 0.48 burts/s.

Logo após as sinapses serem ligadas em *t ~ 10.2s*, um ritmo periódico trifásico com frequência de oscilação ~ *0.48 burts/s* e bastante simétrico é observado (aqui chamamos de simetria a semelhança entre os sianais dos ENs, todos parecidos diferindo apenas em uma fase). No exemplo mostrado na Figura 35 a alternância de fases ocorre

de acordo com: EN0-EN1-EN2-EN0...

Diferente do que ocorre em anéis de inibição unidirecionais, como o mostrado anteriormente na Figura 27, em anéis com mútua inibição a simetria da topologia implica na possibilidade da presença de biestabilidade na fase dos disparos. Para ilustrar esta característica repetimos o experimento com a topologia simétrica mostrada na Figura 34 mas agora desligando e ligando novamente as sinapses de tempos em tempos para observar como o rítmo é restabelecido em cada caso. Na Figura 36 mostramos um trecho deste experimento em que ocorreu a troca da ordem dos disparos claramente de acordo com a previsão da biestabilidade. No trecho mostrado, ocorre a sequência EN0-EN1-EN2-EN0 até *t ~ 22.6 s,* quando o RTLDC é desativado. Ligando novamente as correntes em *t ~ 23,8 s* o mesmo ritmo oscilatório é restabelecido mas a ordem de disparo mudou para EN0-EN2-EN1-EN0.

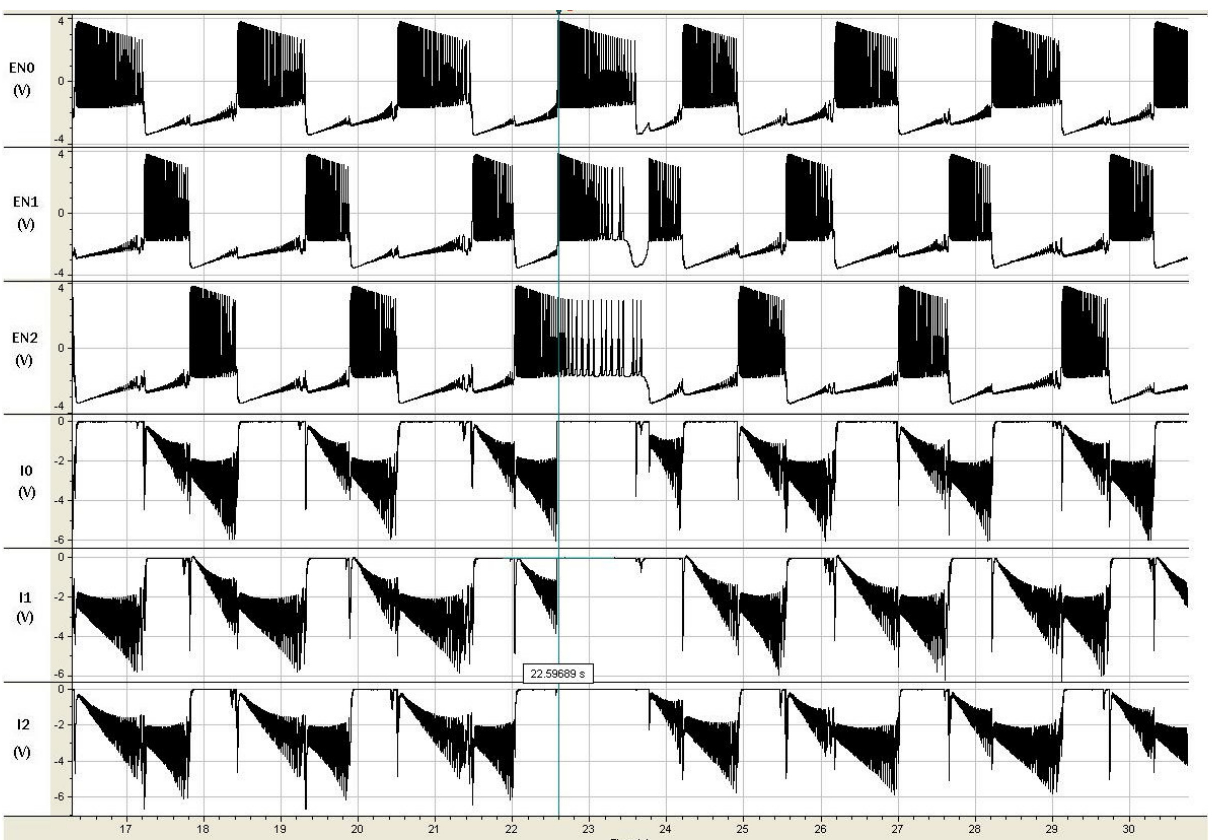


Figura 37: Potenciais de membrana de três neurônios com mútua inibição em anel e suas respectivas correntes. O RTLDC é desligado em *t ~22,6 s* e religado em *t ~ 23,8 s* , quando se observa a inversão da ordem de disparo dos ENs, evidenciando a biestabilidade presente em uma topologia do tipo anel com mútua inibição entre os neurônios devido à simetria das conexões.

8.2 - CPGs artificiais com 4 neurônios

O primeiro experimento que fizemos com 4 neurônios foi conectá-los com mútua inibição dois a dois conforme mostrado na Figura 37. As sinapses são todas idênticas e *G = 2 nS*. O resultado é mostrado na Figura 38.

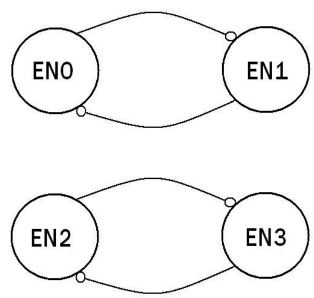


Figura 38: Neurônios conectados dois a dois com mutua inibição.

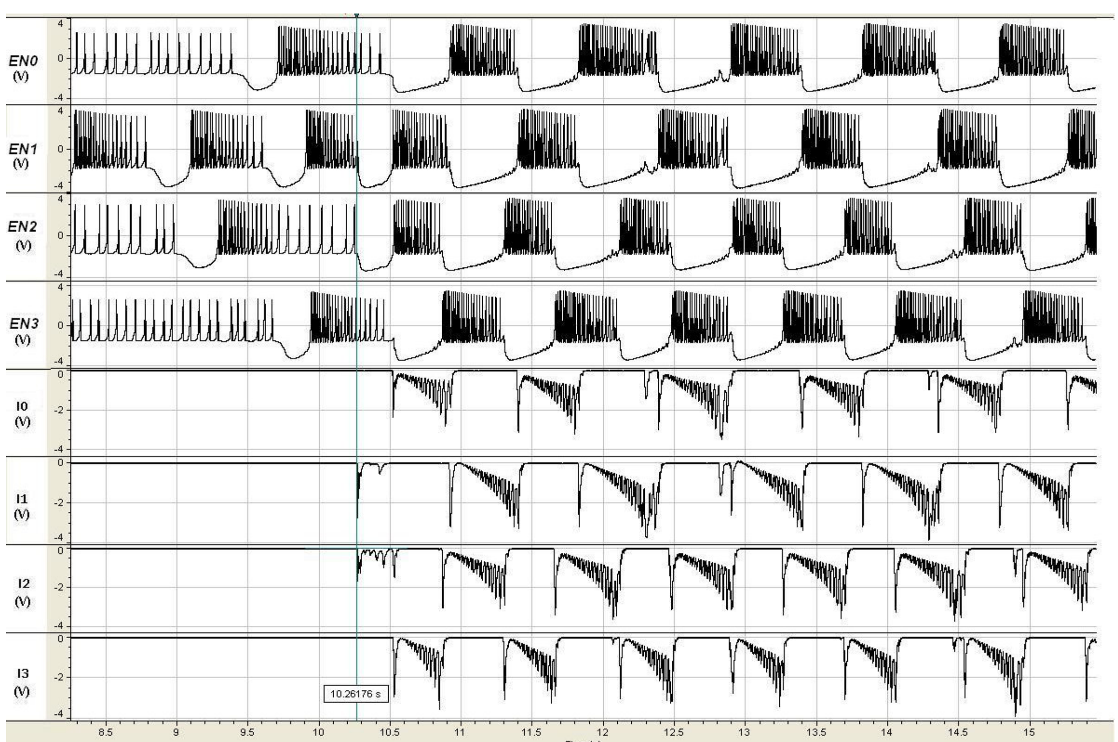


Figura 39: Potenciais de membrana de 4 neurônios eletrônicos e correntes sinápticas produzidas pelo RTLDC quando os neurônios são conectados 2 a 2 com mútua inibição. Inicialmente as sinapses estão desligadas e o comportamento dos neurônios é independente e caótico. Em *t ~ 10.3s* as sinapses são ligadas e um ritmo em antifase é estabelecido em cada par de neurônios. Entretanto o comportamento de um par é independente do outro e as frequências de oscilação são distintas apesar de próximas (~ *1.1 bursts/s* para EN0-EN1 e ~ *1.25 bursts/s* para EN2-EN3).

Inicialmente os Ens estão isolados e comportam-se do modo independente e caótico, como foram ajustados. A partir de *t ~ 10.3s* quatro sinapses idênticas com *G = 2 nS* são ligadas, conectando-os dois a dois com mutua inibição, de acordo com a topologia mostrada na Figura 37, e um rítmo em antifase (alternância de atividade) é estabelecido em cada par de neurônios. O comportamento de um par de neurônios é independente do outro, cada par oscila com uma frequência distinta: ~ *1.1 bursts/s* para EN0-EN1 e ~ *1.25 bursts/s* para EN2-EN3. Estas pequenas diferenças entre os pares de neurônios evidenciam o ajuste intrínseco não idêntico dos ENs feito com o propósito de simular as condições experimentais realistas, já que dificilmente dois neurônios biológicos se comportam exatamente da mesma forma quando isolados.

De maneira semelhante ao que fizemos na seção 8.1.1 para anéis com inibição unidirecional, estudamos como o valor da condutância das sinapses influencia a frequência de oscilação de circuito com mútua inibição. Para isso, repetimos o experimento esquematizado na Figura 37, porém desta vez usando sinapses com *G = 1 nS* no par EN0-EN1 e sinapses com *G = 4 nS* no par EN2-EN3. Neste caso a frequência de oscilação estabelecida nos pares é bem distinta (*~1.7 bursts/s* para EN0-EN1, *~0.8 bursts/s* para EN2-EN3), conforme mostrado na Figura 39.

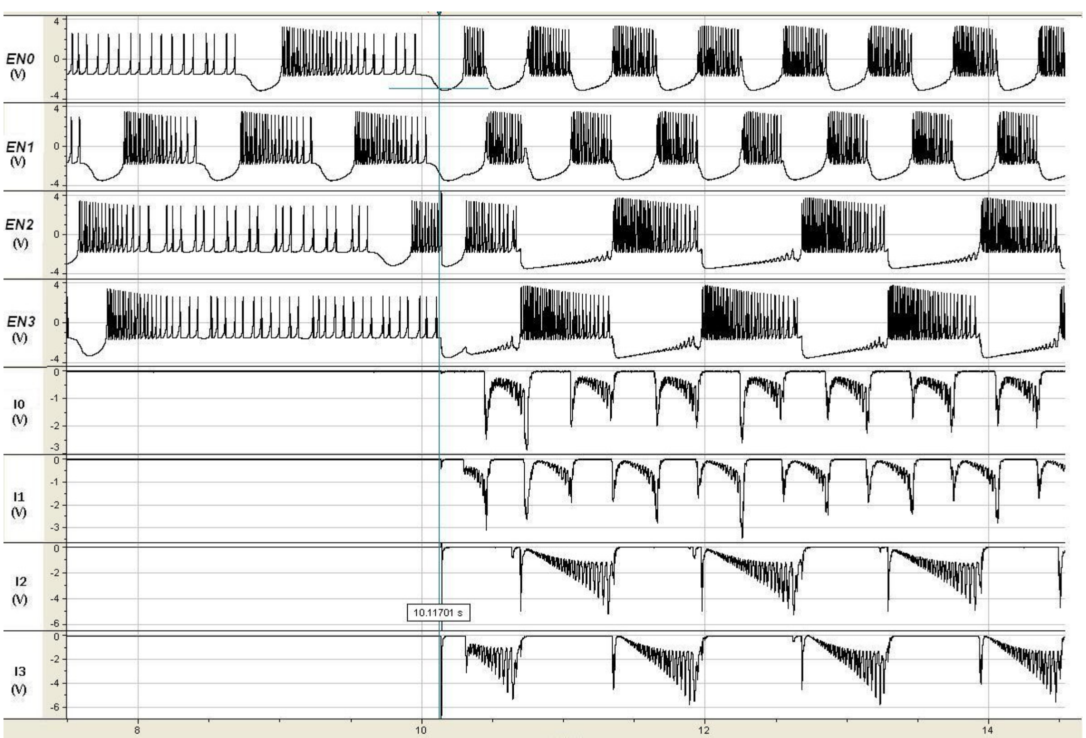


Figura 40: Mudança do comportamento oscilatório dos neurônios quando o par EN0-EN1 é conectado com condutância *G = 1nS* e o par EN2-EN3 com *G = 4nS*. Uma frequência de *~1.7 bursts/s* é estabelecida no par EN0-EN1, enquanto o par EN2-EN3 apresenta frequência de *~0.8 bursts/s* logo após as sinapses serem ligadas em *t ~ 10.1s*.

Também aproveitamos nossa implementação do RTLDC para estudar algumas estratégias observadas em CPGs biológicos para a sincronização de pares de neurônios com mútua inibição (*half-center oscillators*). Uma das estratégias observadas consiste em conectar um neurônio de cada par com mútua inibição, estabelecendo um terceiro *half-center oscillator* de acordo com a topologia mostrada na Figura 40 (EN0-EN2).

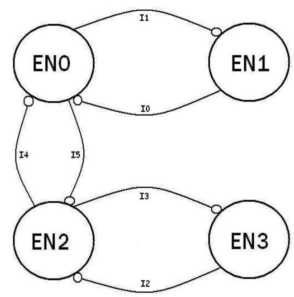


Figura 41: Esquema de ligação em que dois *half-center oscillators* (EN0-EN1 e EN2-EN3) são conectados por mutua inibição não simétrica, um neurônio de cada par é escolhido e um novo *half-center* (EN0-EN2) é estabelecido. Neste experimento, em todas as sinapses, *G = 4 nS*.

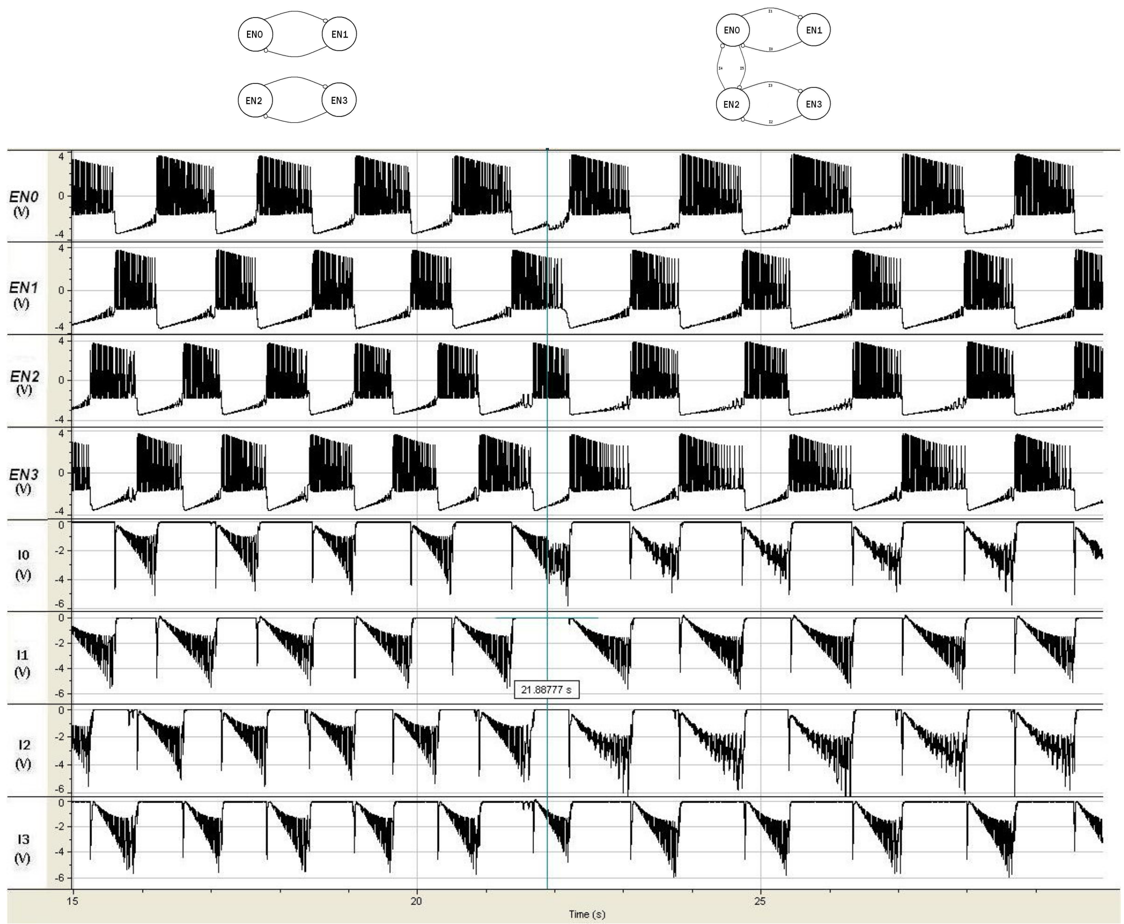


Figura 42: Sincronização de *half-center oscillators*. Até *t ~ 21,9 s* os pares de neurônios oscilam de maneira independente, apenas as sinapses que implementam a mutua inibição entre EN0-EN1 e EN2-EN3 estão ativas. O par EN0-EN1 oscila com frequência ~ *0.7 bursts/s* e o par EN2-EN3 oscila com frequência ~ *0.8 bursts/s*. Quando as sinapses que conectam EN0-EN2 são ligadas em *t ~ 21,9 s* todos os ENs sincronizam quase instantaneamente e um ritmo com frequência menor (~ *0.6 bursts/s*) se estabelece.

Na Figura 41 mostramos o resultado obtido quando ligamos as sinapses que conectam EN0-EN2, ligando os dois *half-center oscillators* que, até *t ~ 21,9 s,* apresentavam comportamento independente (EN0-EN1 oscilava com frequência ~ *0.7 bursts/s* e EN2-EN3 oscilava com frequência ~ *0.8 bursts/s*). Os ENs sincronizam praticamente de modo instantâneo e um ritmo com frequência menor (~ *0.6 bursts/s*) do que era observado nos *half-centers* isolados se estabelece. Este resultado também é observado nos experimentos anteriores: sempre que acrescentamos uma nova sinapse inibitória ou aumentamos a condutância de uma sinapse inibitória já existente a frequência de oscilação diminui em circuitos com mútua inibição. Intuitivamente, este resultado é importante para a sincronização já que, ao conectar (apenas com inibição) dois circuitos funcionando com frequências ligeiramente diferentes, não haveria como fazer o circuito mais lento acelerar seu ritmo para que uma frequência intermediária pudesse ser estabelecida. É interessante observar que EN0 e EN3 (EN1 e EN2) apresentam comportamentos idênticos sem que haja, entretanto, nenhuma conexão direta entre eles.

A segunda estratégia de sincronização observada em alguns CPGs biológicos consiste em ligar um dos neurônios de um *half-center oscillator* a um neurônio do outro através de uma única sinapse inibidora e fazer o inverso com o outro neurônio como mostrado na Figura 42.

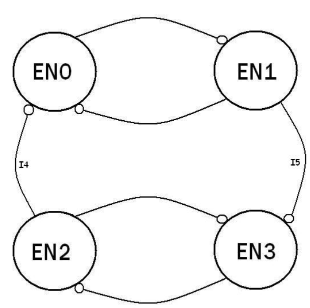
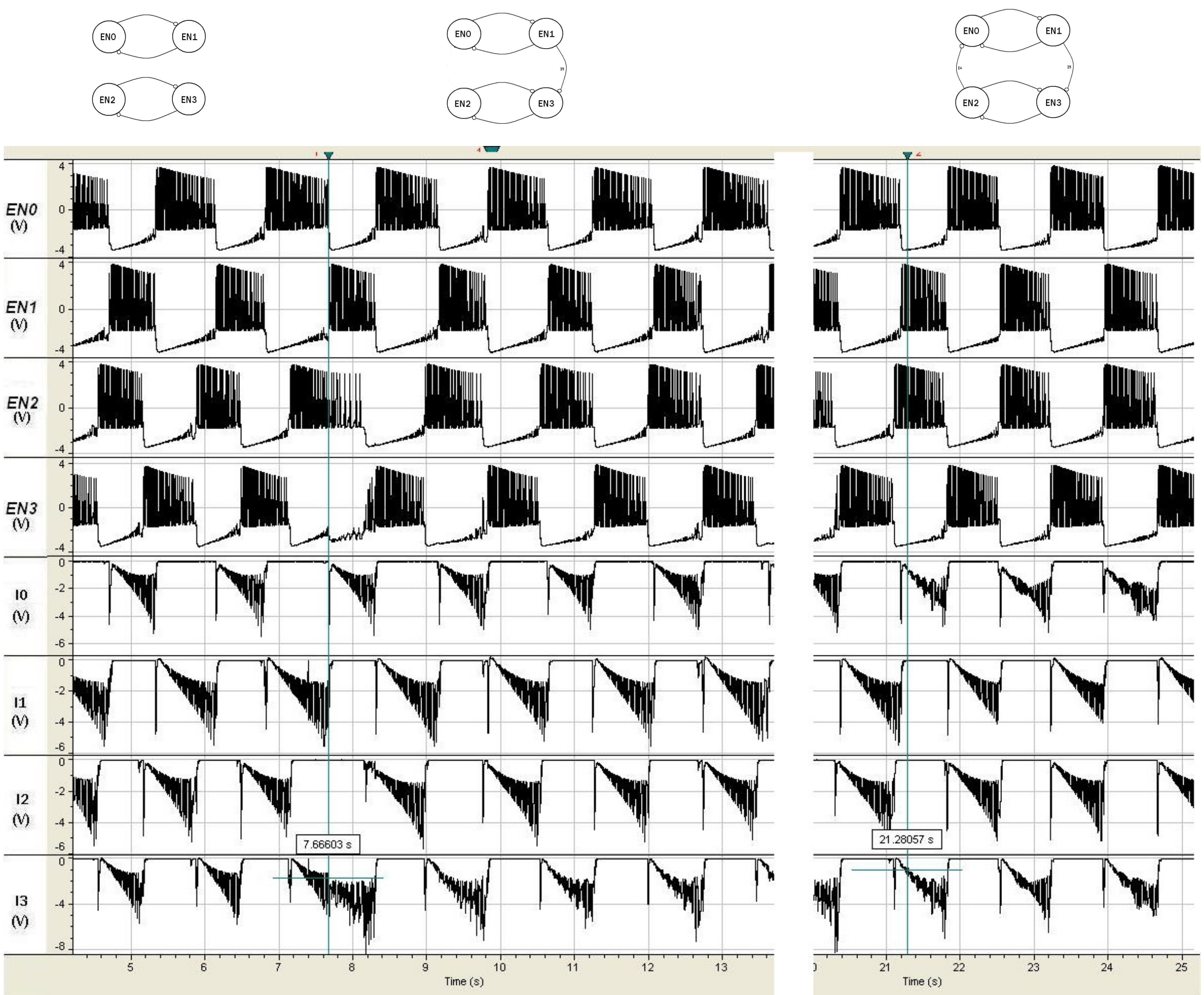


Figura 43: Esquema de ligação em que dois *half-center oscillators* (EN0-EN1 e EN2-EN3) são conectados por mutua inibição simétrica. Um neurônio (EN2) de um *half-center oscillator* inibe um neurônio do outro (EN0) e uma sinapse simétrica de realimentação é estabelecida nos outros dois neurônios (de EN1 para EN3), sem que um novo *half-center* seja estabelecido. Em todas as sinapses *G = 4 nS.*

Na Figura 43(a) apresentamos inicialmente os dois osciladores (EN0-EN1 e EN2-EN3) funcionando independentemente (c/ frequências ~ *0.67* e *0.7 bursts/s*, respectivamente). Em t ~ 7.7 s é ligada a sinapse inibidora de EN1 para EN3, e os dois osciladores passam a operar de modo sincronizado com uma frequência de ~ 0.67 bursts/s. Entretanto, refletindo a assimetria da topologia, EN2 dispara um pouco antes de EN1 e a duração do burst de EN0 é ligeiramente maior que a de EN3. Em 43(b), ao ligarmos a sinapse inibidora de EN2 para EN0 (t ~ 21.3 s) e estabelecermos um cicuito idêntico ao diagrama mostrado na Figura 42, ocorre a sincronização completa entre os ENs e a frequência de oscilação estabelecida é de ~ 0.7 bursts/s. Todas as sinapses tem G = 4 nS.



(a) **(b)**

Figura 44: **(a)** Inicialmente os dois half-center oscillators (EN0-EN1 e EN2-EN3) estão funcionando independentemente (c/ frequências ~ *0.67 bursts/s* e ~ *0.7 bursts/s*, respectivamente). Em t ~ 7.7 s é ligada uma sinapse inibidora de EN1 para EN3, e os dois osciladores passam a operar de modo sincronizado com uma frequência ligeiramente maior (~ *0.67 bursts/s*) entretanto EN2 dispara um pouco antes de EN1 e a duração do burst de EN0 é ligeiramente maior que a de EN3. Em **(b),** ao ligarmos a sinapse inibidora de EN2 para EN0 (*t ~ 21.3 s*) ocorre a sincronização completa entre os ENs e a frequência de oscilação estabelecida é de ~ 0.7 bursts/s. Todas as sinapses tem *G = 4 nS.*

8.3 - Neurônio artificial tipo Hodgkin-Huxley Estocástico

Nesta seção descrevemos os resultados preliminares obtidos com a implementação de um modelo de neurônio estomatogástrico do tipo Hodgkin-Huxley Estocástico. O modelo implementado no RTLDC tem dois compartimentos, conforme descrito no Capítulo 5 e foi desenvolvido por Carelli e colaboradores como uma ampliação de seu trabalho anterior (Carelli et al., 2005).

O modelo consiste em uma versão estocástica de um modelo determinístico, descrito na seção 3.4, que é baseado em medidas eletrofisiológicas das condutâncias iônicas de neurônios do STG. Também implementamos duas sinapses químicas no modelo estocástico para podermos conectá-lo com neurônios artificiais ou biológicos: em uma das sinapses o modelo estocástico é o neurônio pré-sináptico e na outra ele é o neurônio pós-sináptico. Uma imagem congelada da tela do computador com o RTLDC mostrando a interface gráfica elaborada para o modelo estocástico e suas sinapses são mostradas na Figura 44, onde podem ser vistos os valores de parâmetros selecionados.

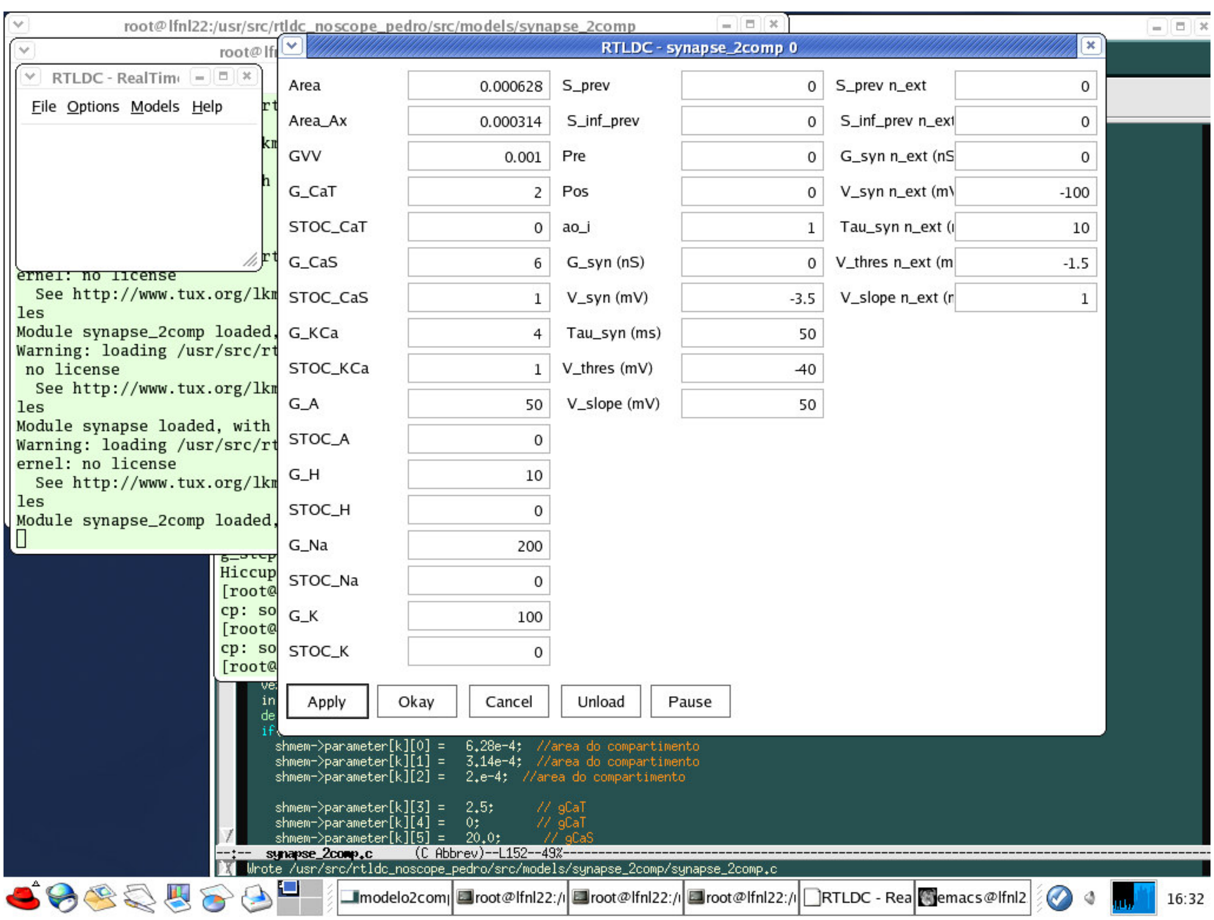


Figura 45: Imagem da tela congelada do computador com o RTLDC e a interface gráfica do neurônio modelo estocástico e suas sinapses químicas.

O modelo implementado por Carelli e colaboradores permite selecionar a natureza do comportamento individual de cada uma das correntes iônicas do modelo: se é do tipo determinístico, onde equações diferenciais são integradas em tempo real, ou se são do tipo estocástico devido às probabilidades de transição entre diferentes estados em uma cadeia Markoviana que representa o comportamento de cada canal iônico (Schneidman, Freedman & Segev, 1998; Carelli et al., 2005).

Para ilustrar o comportamento do modelo tipo Hodgkin-Huxley estocástico de um neurônio do STG nosso programa foi inicialmente ajustado para que todas as correntes fossem determinísticas como no modelo HH original (descrito na seção 3.4). Durante a execução do programa, em *t ~ 20 s*, a corrente lenta de cálcio (Ca_S) responsável por iniciar o platô de burst e a corrente lenta de potássio dependente da concentração de cálcio intracelular $K_{[Ca]}$ que termina o platô tiveram seu comportamento mudado de determinístico para estocástico, produzindo a série temporal mostrada na Figura 45.

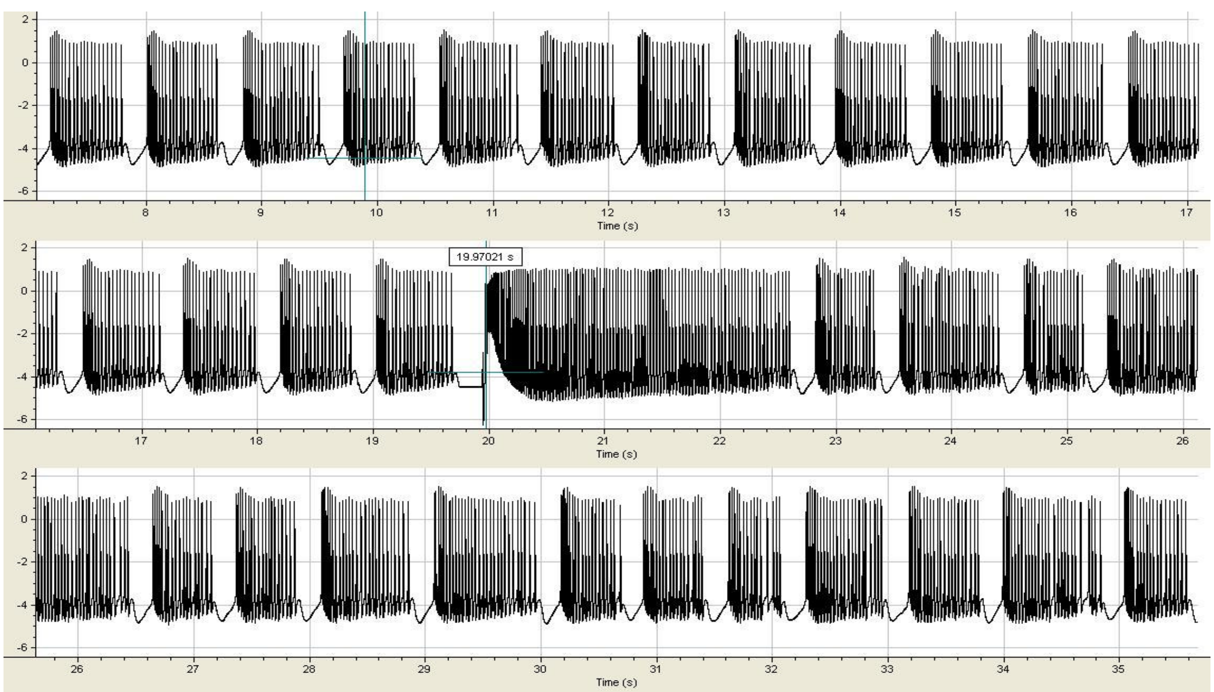


Figura 46: Implementação no RTLDC de um modelo tipo Hodkin-Huxley de um neurônio estomatogástrico de crustáceo – o eixo *y* representa o potencial de membrana do neurônio modelo estocástico produzido pelo RTLDC (em mV × 100). Até *t ~ 20 s* as correntes iônicas são computadas de modo determinístico, integrando em tempo real as equações diferenciais que descrevem o comportamento das correntes. Em *t ~ 20 s* duas das sete correntes iônicas do modelo - Ca_S (responsável por iniciar o platô de burst) e $K_{[Ca]}$ (que termina o platô) tem seu comportamento alterado para estocástico. Assim que as correntes estocásticas são ativadas a periodicidade dos bursts desaparece, estabelecendo-se um comportamento irregular com duração dos bursts e número de spikes/burst aleatórios semelhantes aos encontrados em neurônios biológicos isolados do STG (Selverston et al., 2000).

Assim que as correntes estocásticas são ativadas o comportamento periódico do modelo HH desaparece e um comportamento irregular com duração dos bursts e número de spikes/burst aleatórios, semelhante ao encontrado em neurônios biológicos isolados do STG (Selverston et al., 2000), é estabelecido, confirmando em um modelo de 2 compartimentos em RTLDC os resultados obtidos nas simulações numéricas de Carelli e colaboradores (Carelli et al., 2005).

Para ilustrar o comportamento de um CPG em que um dos neurônios é um modelo do tipo HH estocástico conectamos este a um neurônio eletrônico HR intrinsecamente caótico formando um par com mútua inibição – um *half-center oscillator*. O resultado é mostrado na Figura 46 onde **IN4** é um sinal elétrico produzido pelo RTLDC para representar o potencial de membrana do neurônio modelo estocástico simulado em tempo real (mostrado em mV × 100 para otimizar a aquisição de dados); **IN0** é o potencial de membrana de um neurônio eletrônico HR ; **IN5** é o sinal elétrico de corrente produzido pelo RTLDC para inibir o EN quando o modelo HHestocástico dispara e **IN6** é um sinal elétrico produzido pelo RTLDC que é proporcional à corrente computada para inibir internamente o HHestocástico quando o EN dispara (mostrada em nA ÷ 10). Observe que o RTLDC produz 3 sinais elétricos, assim esse experimento não seria possível sem o demultiplex analógico.

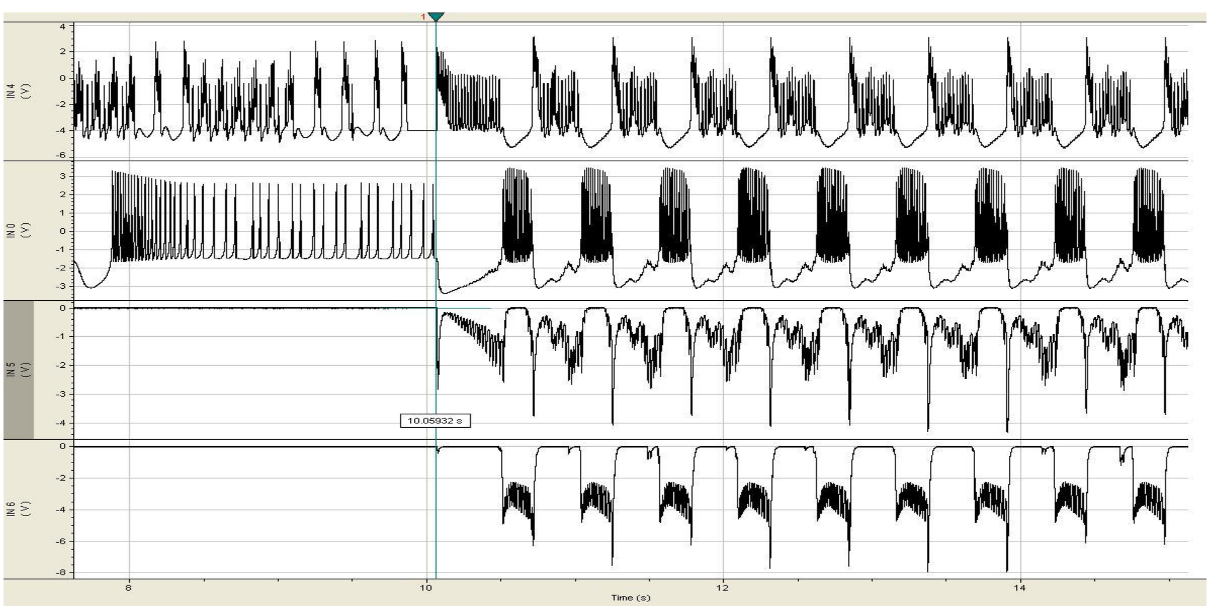


Figura 47: Neurônio modelo HH estocástico conectado a um neurônio eletrônico HR com mútua inibição, formando um *half-center oscillator*. IN4 representar o potencial de membrana do neurônio modelo estocástico produzido pelo RTLDC (em mV × 100); IN0 é o potencial do EN; IN5 é a corrente produzida pelo RTLDC para o HHestocástico inibir o EN e IN6 é proporcional à corrente computada (em nA ÷ 10) para inibir internamente o HHestocástico quando o EN dispara. No início o HHestocástico está ativo no RTLDC mas as sinapses estão desligadas e os neurônios se comportam de modo irregular e independente. Assim que as sinapses são ligadas em $t \sim 10\ s$ um ritmo periódico de oscilação é produzido.

O resultado é bastante não intuitivo, uma vez que um dos neurônios se comporta isoladamente de modo caótico e o outro é estocástico, mas, quando conectados, um ritmo periódico de oscilação é produzido. Isto demonstra claramente que o ritmo é uma propriedade emergente de um sistema complexo que não pode ser entendido simplesmente como a superposição dos comportamentos individuais de seus componentes.

9. Conclusão

Descrevemos a implementação de um protocolo de Dynamic Clamp e de um circuito eletrônico auxiliar demultiplex analógico para a produção de sinapses e condutâncias artificiais em neurônios biológicos/artificiais utilizando um computador pessoal. Conseguimos contornar algumas limitações de muitos programas de Dynamic Clamp comerciais de custo mais elevado: roda em tempo real em uma plataforma Linux Real-Time, funciona com uma placa de aquisição de dados ADC/DAC comercial comum, pode conectar até oito neurônios (através do circuito demultiplex analógico) e ainda atinge frequências de atualização da corrente de até 3 kHz (para 8 correntes de saída).

Apresentamos os resultados obtidos com diversos testes de funcionalidade que fizemos, usando o programa adaptado e o circuito demultiplex para produzir sinapses em tempo real e conectar diversos neurônios artificiais tipo HR em pequenas redes com três e quatro neurônios. Reproduzimos vários resultados observados experimentalmente e em simulações numéricas de CPGs como: a importância da presença de mútua inibição entre pares de neurônios para produzir rítmos com frequência que dependa da condutância das sinapses e para a sincronização dos neurônios de um CPG. Também mostramos alguns resultados preliminares obtidos com a primeira implementação de um modelo de neurônio estocástico tipo Hodgkin-Huxley em um programa de Dynamic Clamp e a notável capacidade que um sistema neural complexo com apenas 2 neurônios como o mais simples CPG tem de produzir rítmos periódicos a partir de componentes intrinsecamente irregulares.

Estes são apenas alguns exemplos do potencial de nossa implementação do RTLDC com demultiplex analógico e 8 canais de saída. Mesmo assim, já pudemos observar uma enorme riqueza de detalhes e capacidades do programa que podem ser usadas para explorar muitos outros aspectos e questões relativas à geração de padrões por redes neurais biológicas, artificiais ou híbridas. Entretanto tais explorações são objetivos futuros e fogem do escopo deste trabalho.

Com o aparecimento de computadores e placas de aquisição de dados cada vez mais rápidos, a tendência é que em pouco tempo a limitação mais séria que restará aos protocolos de Dynamic Clamp será apenas a de origem eletrofisiológica - o fato de toda a “corrente sináptica” ser injetada no soma dos neurônios biológicos e estes não se comportarem na maior parte das vezes como compartimentos únicos isotônicos.

10. Referências

Ayers, J. (2004) Underwater walking, Arthropod structure & development 33, 347-360.

Bear F. Mark, Paradiso A. Michel, Connors W. Barry (2002): Neurociencia Desvendando o Sistema Nervoso, 2rd ed., Artmed.

Butera, R. J.,Wilson, C.G., DelNegro, C.A. and Smith, J. C.(2001): A methodology for achieving high-speed rates for artificial conductance injection in electrically excitable biological cells. IEEE Trans. Biomed. Eng. 48, 1460-1470.

Carelli, P. V., Reyes, M. B., Sartorelli, J. C. and Pinto, R. D. (2005): Whole cell stochastic model reproduces the irregularities found in the membrane potential of bursting neurons, J. Neurophysiol. 94, 1169-1179.

Dayan, P. and Abbot, L.F. (2001): Theoretical neuroscience: Computational and mathematical modeling of neural systems. MIT Press, Cambridge, MA.

Dorval, A. D., Christini, D. J., White, J. A. (2001): Real-Time Linux Dynamic Clamp: A Fast and Flexible Way to Construct Virtual Ion Channels in Living Cells. Annals of Biomedical Engineering 29, 897.

Falcke M., Huerta R., Rabinovich M. I., Abarbanel H. D. I., Elson, R. and Selverston A. I. (2000): Modeling observed chaotic oscillations in bursting neurons: the role of calcium dynamics and IP3, Biol. Cybern. 82, 517-527.

Fitzhugh, R. (1961): Impulses and physiological states in theoretical models of nerve membrane. Biophys. J. 1, 445-466.

Hille, B. (2001): Ionic Channels of Excitable Membranes, 3rd ed., Sinaur, Sunderland. MA.

Hindmarsh, J.L. and Rose, R.M. (1982), A model of the nerveimpulse using two first-order differential equations, Nature 296, 162.

Hindmarsh, J. L. and Rose, R. M. (1984): A model of neuronal bursting using three coupled first order differential equations, Proc. R. Soc. Lond. B. 221, 87-102.

Hodgkin, A. L. and Huxley, A. F. (1952): A quantitative description of membrane current and its application to conduction and excitation in nerve. J. Physiol. Lond. 117, 500-54.

Holden, A. V. (1997): Nonlinear Science - The impact of Biology, Int. J. Bif. and Chaos 7, 2075-2104

Izhikevich, E. (2000): Neural excitability, spiking and bursting, Int. J. Bif. Chaos 10, 1171-1266.

Kandel E.R., Schwartz J.H., e Jessel T.M. (1991): Principles of Neural Science, 3rd ed., Appleton & Lange, Connecticut.

Koch, C. and Segev, I. (eds.) (1998): Methods in Neuronal Modeling: from ions to networks, 2nd ed., MIT Press, Cambridge, MA.

Levitan, I.B., Kaczmareck, L.K. (1997): The Neuron: cell and molecular biology, Oxford University Press.

Mainen, Z. F. and Sejnowski, T. J. (1995): Reliability of spike timing in neocortical neurons. Science 268, 1503-8.

Marder, E. (1998): From biophysics to models of network function, Annu. Rev. Neurosci. 21, 25-45.

Mulloney, B., Selverston, A. I. (1974), Organization of the stomatogastric ganglion of the spiny lobster, J. Comp. Physiol. 91, 1-32.

Nicolelis, M. A. L. (2003) Brain machine interfaces to restore motor function and probe neural circuits, Nature Rev. Neurosci. 4, 417-422.

Pinto, R.D., Varona, P., Volkovskii, A.R., Szücs A., Abarbanel H.D.I. e Rabinovich M.I. (2000): Synchronous behavior of two coupled electronic neurons, Phys. Rev. E 62, 2644.

Pinto, R. D., Elson, R. C., Szücs, A., Rabinovich, M. I., Selverston A. I., and Abarbanel, H. D. I.(2001): Extended dynamic clamp: controlling up to four neurons using a single desktop computer and interface, J. Neurosci. Methods 108, 39-48.

Prinz, A.A., Thirumalai, V. and Marder, E.(2003): The functional consequences of changes in the strength and duration of synaptic inputs to oscillatory Neurons, J. Neurosci. 23, 943-954.

Prinz, A. A. (2004): Neural networks: models and neurons show hybrid vigor in real time. Curr. Biol. 14, R661-R662.

Prinz, A. A., Abbott, L. F., Marder, E. (2004): The Dynamic Clamp comes of age. TRENDS Neurosci. 27, 218-224.

Rieke, F., Warland, D., van Steveninck, R. R. e Bialek, W. (1997): Spikes - Exploring the neural code, MIT press, London, England.

Rinzel, J. e Ermentrout, G. B. (1998): Analysis of neural excitability and oscillations. Em: Methods in Neuronal Modelling, eds. C. Koch e I. Segev, MIT Press, Cambridge, MA.

Schmidt-Nielsen K. (1996): Fisiologia animal, 5ª ed., Santos Editora, São Paulo.

Schneidman, E., Freedman, B., Segev,I. (1998): Ion channel stochasticity may be critical in determining the reliability and precision of spike timing, Neural Comput. 10, 1679-1703.

Selverston, A. I. and Moulins, M., eds. (1986) The crustacean stomatogastric system, Springer-Verlag, Berlim.

Selverston, A. I., Rabinovich, M. I., Abarbanel, H. D. I., Elson, R. C., Szücs, A., Pinto, R. D., Huerta, R., and Varona P. (2000): Reliable Circuits from Irregular Neurons: A Dynamical Approach to Understanding Central Pattern Generators, J. Physiol. (Paris) 94, 357-74.

Sharp, A. A., Abbot, L. F. e Marder, E. (1992): Artificial Electrical Synapses in Oscillatory Networks, J. Neurophysiol. 67, 1691-1694.

Sharp, A. A., O'Neil, M. B., Abbot, L. F. e Marder, E. (1993): Dynamic clamp: computer-generated conductances in real neurons, J. Neurophysiol. 69, 992-995.

Sharp, A. A., Skinner, F. K., e Marder, E. (1996): Mechanisms of Oscillation in Dynamic Clamp Constructed Two-Cell Half-Center Circuits, J. Neurophysiol. 76, 867-883.

Szücs, A., Varona, P., Volkovskii, A. R., Abarbanel, H. D. I., Rabinovich, M. I. e Selverston, A. I. (2000): Interacting biological and electronic neurons generate realistic oscillatory rhytms, Neuroreport 11, 563-569.

APÊNDICE 1 – daq_spec.h

Arquivo modificado por Rogério Mazur, leibnis@yahoo.com.br, a partir da versão original de Jonathan Bettencourt, jbetten@bu.edu, http://bme.bu.edu/ndl/rtldc.html, em 06/2005 para controlar um cicuito demultiplexador analógico usado para produzir até 8 sinais analógicos de corrente a partir de placas DAQ com apenas das 2 saídas analógicas. As alterações/inclusões estão alinhadas mais à esquerda, enquanto as partes originais do programa encontram-se alinhadas mais à direita. Uma cópia da licença de uso/cópia/modificação do programa (GNU General Public License) encontra-se no **Apêndice 3**.

daq_spec.h

```
            // Copyright (C) 2003, 2004 by Jonathan Bettencourt (the original daq_spec.h)
            // Copyright (C) 2006 by Rogério Mazur - leibnis@yahoo.com.br
            // (modifications for including an analog demultiplex)
            // This program is free software; you can redistribute it and/or modify
            // it under the terms of the GNU General Public License as published by
            // the Free Software Foundation; either version 2 of the License, or
            // any later version.
            // This program is distributed in the hope that it will be useful,
            // but WITHOUT ANY WARRANTY; without even the implied warranty of
            // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
            // GNU General Public License for more details.
            // You should have received a copy of the GNU General Public License along
            // with this program; if not, write to the Free Software Foundation, Inc.,
            // 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

            #ifndef __DAQ_SPEC_H
            #define __DAQ_SPEC_H
            #define AI_SUBDEV_NUMBER      0
            #define MAX_AI_CHANNELS                 16
            #define NUM_AI_RANGES                   16
            #define AI_INT_MAX                4095
            #define AI_INT_ZERO      {2048,2048,2048,2048,2048,2048,2048,2048,0,0,0,0,0,0,0,0}
            #define AI_FLOAT_MAX  {10,5,2.5,1,0.5,0.25,0.1,0.05,20,10,5,2,1,0.5,0.2,0.1}
            #define AI_FLOAT_MIN   {-10,-5,-2.5,-1,-0.5,-0.25,-0.1,-0.05,0,0,0,0,0,0,0,0}
            #define AO_SUBDEV_NUMBER      1

#define REAL_MAX_AO_CHANNELS    2 /* Numero real de canais */
#define MAX_AO_CHANNELS                 8 /* Numero de canais multiplexados */

            #define NUM_AO_RANGES                   4
            #define AO_INT_MAX                4095
            #define AO_INT_ZERO      {2048,2048,2048,2048}
            #define AO_FLOAT_MAX {10,5,2.5,1}
            #define AO_FLOAT_MIN  {-10,-5,-2.5,-1}

#define DO_SUBDEV_NUMBER        2
#define MAX_DO_CHANNELS        8
#define NUM_DO_RANGES          1

            #endif
```

APÊNDICE 2 – rtldc_demultiplex.c

Programa modificado a partir da versão original rtldc.c de Jonathan Bettencourt, jbetten@bu.edu, http://bme.bu.edu/ndl/rtldc.html, em 06/2005 por Rogério Mazur, leibnis@yahoo.com.br, para controlar um cicuito demultiplexador analógico usado para produzir até 8 sinais analógicos de corrente a partir de placas DAQ com apenas das 2 saídas analógicas. As alterações/inclusões estão alinhadas mais à esquerda, enquanto as partes originais do programa encontram-se alinhadas mais à direita. Uma cópia da licença de uso/cópia/modificação do programa (GNU General Public License) encontra-se no **Apêndice 3**.

rtldc_demultiplex.c

```
// Copyright (C) 2003, 2004 by Jonathan Bettencourt (the original rtldc.c)
// Copyright (C) 2006 by Rogério Mazur - leibnis@yahoo.com.br
// (modifications for including an analog demultiplex)
// This program is free software; you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation; either version 2 of the License, or
// any later version.

// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.

// You should have received a copy of the GNU General Public License along
// with this program; if not, write to the Free Software Foundation, Inc.,
// 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.



// ***** System Headers ***** //

#include <settings.h>

#if defined RTLINUX
#define __RTL__
#define __KERNEL__
#define MODULE
#define MODVERSIONS
#include <rtl.h>
#include <mbuff.h>
#include <pthread.h>
#elif defined RTAI
#define __KERNEL__
#define MODULE
#include <linux/module.h>
#include <rtai.h>
#include <rtai_sched.h>
#include <rtai_shm.h>
```

```
#endif

#include <asm/string.h>
#include <linux/comedi.h>
#include <linux/comedilib.h>
#include <rtldc.h>
#include <module.h>
#include <shmem.h>
#include <daq_spec.h>

#ifdef MODULE_LICENSE
MODULE_LICENSE("GPL");
#endif

MODULE_AUTHOR("Jonathan Bettencourt <jbetten@bu.edu>");
MODULE_DESCRIPTION("RTLDC Real-Time Control Engine
<http://bme.bu.edu/ndl/rtldc.html>");

EXPORT_NO_SYMBOLS;

#define DEBUG_MODE 1
// ***** Global Variables ***** //

  // *** Shared Memory *** //
volatile rtldc_shm *shmem;                              // Shared Memory Pointer

  // *** Real Time Thread *** //
#if defined RTLINUX
pthread_t       rt_thread;                        // RTLinux Thread
#elif defined RTAI
RT_TASK         rt_thread;                        // RTAI Thread
#else
#error "Must #define RTLINUX or RTAI"
#endif

  //*** Comedi Device *** //
#if defined OLD_COMEDI
int             daq_dev = DEVICE_NUMBER;                   // Comedi DAQ Dev
#elif defined NEW_COMEDI
comedi_t        *daq_dev;                         // Comedi DAQ Dev
#else
#error "Must #define OLD_COMEDI or NEW_COMEDI"
#endif

int             ai_subd = AI_SUBDEV_NUMBER;         // Comedi AI Subdevice
int             ao_subd = AO_SUBDEV_NUMBER;         // Comedi AO Subdevice

                                                  int    do_subd = DO_SUBDEV_NUMBER;
                                                   // Comedi DO Subdevice (saída digital)

  // *** Input Channel *** //
lsampl_t        ai_data[MAX_AI_CHANNELS];                  // AI Channel Data
unsigned int    ai_chan[MAX_AI_CHANNELS];                  // AI Channel States
```

```
int          ai_zero[MAX_AI_CHANNELS];              // AI Channel Zeros
int          ai_num;                               // AI Number of Active Channels
int          ai_active[MAX_AI_CHANNELS];            // AI List of Active Channels
double       ai_conversion[MAX_AI_CHANNELS];         // AI Channel
Conversions
comedi_insn  daqi[MAX_AI_CHANNELS];                 // AI Instruction Array

  // *** Output Channel *** //
lsampl_t     ao_data[MAX_AO_CHANNELS];     // AO Channel Data

                                        lsampl_t
                          ao_real_data[REAL_MAX_AO_CHANNEL
                                S]; // Real AO Channels data

unsigned int  ao_chan[MAX_AO_CHANNELS];    // AO Channel States
int          ao_zero[MAX_AO_CHANNELS];     // AO Channel Zeros
int          ao_num;                      // AO Number of Active Channels
int          ao_active[MAX_AO_CHANNELS];   // AO List of Active Channels
double       ao_conversion[MAX_AO_CHANNELS]; // AO Channel Conversions
comedi_insn  daqo[MAX_AO_CHANNELS];        // AO Instruction Array

// *** Digital Output Channel **//
lsampl_t     do_data_insn[2];          // DO Channel data to config
comedi_insn  ddqo[MAX_DO_CHANNELS];     // DO Instruction

  // *** Function Table *** //
function_table  function_info[MAX_FUNCTIONS];          // Table of Function
Information

  // *** Fifo Data *** //
double
sync_fifo_data[MAX_AI_CHANNELS+MAX_AO_CHANNELS+MAX_FUNCTION
S*MAX_STATES];

// ***** Function Prototypes ***** //

  // *** Initialization Functions *** //
int init_module(void);
int init_shmem(void);
int init_comedi(void);
int init_rt_thread(void);
void initialize(void);

  // *** Cleanup Functions *** //
void cleanup_module(void);
void cleanup_shmem(void);
void cleanup_comedi(void);
void cleanup_rt_thread(void);

  // *** Main Function *** //
#if defined RTLINUX
void *rtldc(void *);
#elif defined RTAI
```

```
void rtldc(int param);
#endif

  // *** Helper Functions *** //
void update_channels(void);
void update_fxn(void);
void load_fxn(void);
void unload_fxn(void);
void pause_fxn(void);
void unpause_fxn(void);
void zero_daq_output(void);
```

void zero_ddq_output(void);

```
// ***** Function Code ***** //

int
init_module(void)
{
#ifdef DEBUG_MODE
  printk("init_module : starting\n");
#endif

  if(init_shmem())     goto error_shmem;
  if(init_comedi())    goto error_comedi;
  if(init_rt_thread()) goto error_thread;

  initialize();

#ifdef DEBUG_MODE
  printk("init_module : finished\n");
#endif

  return 0;

 error_thread:
  cleanup_comedi();
 error_comedi:
  cleanup_shmem();
 error_shmem:
  return 1;
}

int
init_shmem(void)
{
  int i,j;

#ifdef DEBUG_MODE
  printk("init_shmem : starting\n");
#endif
```

```
#if defined RTLINUX
  if((shmem = (rtldc_shm *)mbuff_alloc(SHM_ADDR,sizeof(rtldc_shm))) == NULL) {
    printk("Failed to Initialize the Shared Memory\n");
    return 1;
  }
#elif defined RTAI
  if((shmem = (rtldc_shm *)rtai_kmalloc(nam2num(SHM_ADDR),sizeof(rtldc_shm)))
== NULL) {
    printk("Failed to Initialize the Shared Memory\n");
    return 1;
  }
#else
#error "Must #define RTLINUX or RTAI"
#endif

  shmem->record = 0;

  shmem->state_flag = NO_UPDATE;

  shmem->main_fxn = NULL;
  shmem->update_fxn = NULL;
  shmem->fid = -1;

  for(i=0;i<MAX_AI_CHANNELS;i++) {
    shmem->ai_scale[i] = -1;
    shmem->ai_units[i] = 2;
    //    shmem->ai_lead[i]  = AREF_DIFF;
    shmem->ai_lead[i]  = 0x00;
    shmem->ai_gain[i]  = 1;
  }

  for(i=0;i<MAX_AO_CHANNELS;i++) {
    shmem->ao_scale[i] = -1;
    shmem->ao_units[i] = 2;
    //    shmem->ao_lead[i]  = AREF_GROUND;
    shmem->ao_lead[i]  = 0x00;
    shmem->ao_gain[i]  = 1;
  }
  shmem->ai_scale[0]=0;


shmem->sample_period = 330000;

  shmem->dT_ms = (double)(shmem->sample_period/1.0e6);

  for(i=0;i<MAX_PARAMETERS;i++) {
    for(j=0;j<MAX_FUNCTIONS;j++)
      shmem->parameter[j][i] = 0;
    shmem->share_param[i] = 0;
  }

#ifdef DEBUG_MODE
```

```
  printk("init_shmem : finished\n");
#endif

  return 0;
}



int
init_comedi(void)
{
#ifdef DEBUG_MODE
  printk("init_comedi : starting\n");
#endif

#if defined OLD_COMEDI
  if(daq_dev != comedi_open(daq_dev)) {
    printk("Failed to open Comedi Device\n");
    goto error_open;
  }
#elif defined NEW_COMEDI
  if((daq_dev = comedi_open(DEVICE_NAME))==NULL) {
    printk("Failed to open Comedi Device\n");
    goto error_open;
  }
#else
#error "Must #define OLD_COMEDI or NEW_COMEDI"
#endif
  if(comedi_lock(daq_dev,ai_subd)) {
    printk("Failed to lock AI Subdevice\n");
    goto error_aisubd;
  }
  if(comedi_lock(daq_dev,ao_subd)) {
    printk("Failed to lock AO Subdevice\n");
    goto error_aosubd;
  }

/* O sub-dispositivo de Entrada/Saida digital (DIO) e' o 2 */
if ( comedi_lock(daq_dev, do_subd) ) {
  printk("Failed to lock DO subdevice\n");
  goto error_dosubd;
  }

#ifdef DEBUG_MODE
  printk("init_comedi : finished\n");
#endif

  return 0;

error_dosubd:
 comedi_unlock(daq_dev,do_subd);
error_aosubd:
```

```
comedi_unlock(daq_dev,ao_subd);

 error_aisubd:
  comedi_unlock(daq_dev,ai_subd);

error_open:
comedi_close(daq_dev);

  return 1;
}

int
init_rt_thread(void)
{
#if defined RTLINUX
  pthread_attr_t     rt_thread_attr;
  struct sched_param rt_thread_sched;
#endif

#ifdef DEBUG_MODE
  printk("init_rt_thread : starting\n");
#endif

#if defined RTLINUX
  if(pthread_attr_init(&rt_thread_attr)) {
    printk("Failed to Initialize Real Time Thread Attribute\n");
    return 1;
  }
  rt_thread_sched.sched_priority = 2;
  if(pthread_attr_setschedparam(&rt_thread_attr,&rt_thread_sched)) {
    printk("Failed to set Real Time Thread Schedule\n");
    return 1;
  }
  if(pthread_attr_setfp(&rt_thread_attr,1)) {
    printk("Failed to enable Floating Point Operations\n");
    return 1;
  }
  if(pthread_create(&rt_thread,&rt_thread_attr,rtldc, (void *)1)) {
    printk("Failed to Create Real Time Thread\n");
    return 1;
  }

if(pthread_make_periodic_np(rt_thread,clock_gethrtime(CLOCK_RTL_SCHED),shme
m->sample_period)) {
    printk("Failed to Make Thread Periodic\n");
    return 1;
  }
#elif defined RTAI
  if(rt_task_init(&rt_thread,rtldc,0,2000,0,1,0)) {
    printk("Failed to Create Real Time Thread\n");
    return 1;
  }
```

```
  if(rt_task_make_periodic(&rt_thread,rt_get_time(),start_rt_timer(nano2count(shmem-
>sample_period)))) {
    printk("Failed to Make Thread Periodic\n");
    return 1;
  }

#else
#error "Must #define RTLINUX or RTAI"
#endif

#ifdef DEBUG_MODE
  printk("init_rt_thread : finished\n");
#endif

  return 0;
}

lsampl_t mydata;
comedi_insn mydaqo;

void
initialize(void)
{
  int i;

#ifdef DEBUG_MODE
  printk("initialize : starting\n");
#endif

  // *** Initialize the function_info array *** //
  for(i=0;i<MAX_FUNCTIONS;i++) {
    function_info[i].active = 0;
    function_info[i].paused = 0;
    function_info[i].main_fxn = NULL;
    function_info[i].update_fxn = NULL;
  }

  // *** Configuring the digital output channels *** //
  do_data_insn[1] = COMEDI_OUTPUT;
  do_data_insn[0] = 0x00FF;
  for (i=0; i<MAX_DO_CHANNELS; i++) {
    if ( comedi_dio_config(daq_dev, do_subd, i, COMEDI_OUTPUT) != 1) {
      printk("Failed to configure DO subdevice\n");
    }

    ddqo[i].insn     = INSN_CONFIG;
    ddqo[i].subdev   = do_subd;
    ddqo[i].n        = 2;
    ddqo[i].data     = do_data_insn;
    ddqo[i].chanspec = i;
    (void)comedi_do_insn(daq_dev, &(ddqo[i]));
    ddqo[i].insn     = INSN_BITS;
```

```
}

// *** Zero the digital output channels *** //
zero_ddq_output();

// *** Zero the daq output channels *** //
zero_daq_output();
mydaqo.insn     = INSN_WRITE;
mydaqo.subdev   = ao_subd;
mydaqo.n        = 1;
mydaqo.data     = &mydata;

  // *** Initailize channels *** //
  update_channels();

  // *** Signal initialization complete *** //
  shmem->state_flag = LOADED;

#ifdef DEBUG_MODE
  printk("initialize : finished\n");
#endif
}

#if defined RTLINUX
void *
rtldc(void *param)
#elif defined RTAI
void
rtldc(int param)
#else
#error "Must #defined RTLINUX or RTAI"
#endif
{

int i;

  for(;;) {
    switch(shmem->state_flag) {
    case UPDATE_FXN:
      update_fxn();
      break;
    case UPDATE_CHANNELS:
      update_channels();
      break;
    case LOAD_FXN:
      load_fxn();
      break;
    case UNLOAD_FXN:
      unload_fxn();
      break;
    case PAUSE_FXN:
      pause_fxn();
```

```
        break;
      case UNPAUSE_FXN:
        unpause_fxn();
        break;
      case SHUTDOWN:

zero_ddq_output();

        zero_daq_output();
        goto exit;
        break;
      case LOADED:
        break;
      default:
        // *** This function was manually inlined because the compiler was not able to do it
    *** //

        // *** Input Data and Move into Sync Fifo Data Array *** //
        for(i=0;i<MAX_AI_CHANNELS;i++)
          if(ai_active[i]) {
              // printk("Reading channel %d\n",i);
            comedi_do_insn(daq_dev,&daqi[i]);
            sync_fifo_data[i] = ai_conversion[i] * (int)(ai_data[i] - ai_zero[i]);
              //           printk("Finished reading channel %d\n",i);
             }

        // *** Reset all Output Channels and States to Zero *** //
        memset(sync_fifo_data+MAX_AI_CHANNELS,(unsigned
    char)0,sizeof(double)*MAX_AO_CHANNELS);

        // *** Execute all Running Functions *** //
        for(i=0;i<MAX_FUNCTIONS;i++)
            if(function_info[i].active && !function_info[i].paused)
              function_info[i].main_fxn(sync_fifo_data,i);

// *** Convert AO Data to DAQ Units and Output *** //

do_data_insn[0] = 0x00FF;
do_data_insn[1] = 0;
comedi_do_insn(daq_dev, &(ddqo[0]));

for(i=0;i<MAX_AO_CHANNELS;i++) {
ao_data[i] = ao_zero[i] + (sync_fifo_data[MAX_AI_CHANNELS+i] * ao_conversion[i]);
if(ao_data[i] < 0)
ao_data[i] = 0;
else if(ao_data[i] >= AO_INT_MAX)
ao_data[i] = AO_INT_MAX-1;

 if(ao_active[i]) {

 mydata      = ao_data[i];
 mydaqo.chanspec = CR_PACK((i%2), shmem->ao_scale[i], shmem->ao_lead[i]);
```

```
 comedi_do_insn(daq_dev,&mydaqo); // if j even, output in channel 0
                                    // if j odd, output in channel 1
 usleep((unsigned)2);
 do_data_insn[1] = 1 << i;
 usleep((unsigned)5);
 comedi_do_insn(daq_dev, ddqo); //sampling the right channel of SMP04.

 }
}
```

```
    }

#if defined RTLINUX
    pthread_wait_np();
  }

exit:
  return NULL;
}
#elif defined RTAI
    rt_task_wait_period();
  }

exit:
  return;
}
#else
#error "Must #defined RTLINUX or RTAI"
#endif

void
update_channels(void)
{
  int i;

  int    ai_range_zero[NUM_AI_RANGES] = AI_INT_ZERO;
  double ai_float_max[NUM_AI_RANGES] = AI_FLOAT_MAX;
  double ai_float_min[NUM_AI_RANGES] = AI_FLOAT_MIN;

  int    ao_range_zero[NUM_AO_RANGES] = AO_INT_ZERO;
  double ao_float_max[NUM_AO_RANGES] = AO_FLOAT_MAX;
  double ao_float_min[NUM_AO_RANGES] = AO_FLOAT_MIN;

#ifdef DEBUG_MODE
  printk("update_channels : starting\n");
#endif

  for(i=0;i<MAX_AI_CHANNELS;i++)
    if(shmem->ai_scale[i] != -1) {
      ai_zero[i] = ai_range_zero[shmem->ai_scale[i]];
```

```
    ai_conversion[i] = (ai_float_max[shmem->ai_scale[i]] - ai_float_min[shmem-
>ai_scale[i]]) * shmem->ai_gain[i] / AI_INT_MAX;
   }

  for(i=0;i<MAX_AO_CHANNELS;i++)
   if(shmem->ao_scale[i] != -1) {
    ao_zero[i] = ao_range_zero[shmem->ao_scale[i]];
    ao_conversion[i] = AO_INT_MAX / (shmem->ao_gain[i] * (ao_float_max[shmem-
>ao_scale[i]] - ao_float_min[shmem->ao_scale[i]]));
   }

  memset(daqi,0,sizeof(comedi_insn)*MAX_AI_CHANNELS);
  memset(daqo,0,sizeof(comedi_insn)*MAX_AO_CHANNELS);

  // *** Prepare the Comedi Input Instruction Structures *** //
  ai_num = 0;
  for(i=0;i<MAX_AI_CHANNELS;i++) {
   if(shmem->ai_scale[i] != -1) {
    ai_active[i] = 1;

    daqi[i].insn       = INSN_READ;
    daqi[i].subdev     = ai_subd;
    daqi[i].n          = 1;
    daqi[i].data       = ai_data+i;
    daqi[i].chanspec = CR_PACK(i,shmem->ai_scale[i],shmem->ai_lead[i]);
    ai_num++;
   }
   else
    ai_active[i] = 0;
  }

  // *** Prepare the Comedi Ouput Instruction Structures *** //
  ao_num = 0;
  for(i=0;i<MAX_AO_CHANNELS;i++) {
   if(shmem->ao_scale[i] != -1) {
    ao_active[i] = 1;
    // printk(KERN_INFO "ativando canal %d\n", i);

    daqo[ao_num].insn       = INSN_WRITE;
    daqo[ao_num].subdev     = ao_subd;
    daqo[ao_num].n          = 1;
    daqo[ao_num].data       = ao_data+i;
    daqo[ao_num++].chanspec = CR_PACK(i,shmem->ao_scale[i],shmem-
>ao_lead[i]);
   }
   else
    ao_active[i] = 0;
  }

// *** Prepare the Comedi Digital Ouput Instruction Structures *** //
do_data_insn[1] = COMEDI_OUTPUT;
do_data_insn[0] = 0x00FF;
```

```
  for (i=0; i<MAX_DO_CHANNELS; i++) {
    ddqo[i].insn     = INSN_BITS;
    ddqo[i].subdev  = do_subd;
    ddqo[i].n        = 2;
    ddqo[i].data     = do_data_insn;
    ddqo[i].chanspec = i;
  }
shmem->state_flag = NO_UPDATE;

#ifdef DEBUG_MODE
  printk("update_channels : finished\n");
#endif
}


void
update_fxn(void)
{
#ifdef DEBUG_MODE
  printk("update_fxn : starting\n");
#endif

#ifdef DEBUG_MODE
  if(!function_info[shmem->fid].active)
#if defined RTLINUX
    rtl_printf("ERROR : Function %d does not exist\n",shmem->fid);
#elif defined RTAI
  rt_printk("ERROR : Function %d does not exist\n",shmem->fid);
#endif
  else
#endif

  function_info[shmem->fid].update_fxn(sync_fifo_data,shmem->fid,0);

  shmem->fid = -1;
  shmem->state_flag = NO_UPDATE;

#ifdef DEBUG_MODE
  printk("update_fxn : finished\n");
#endif
}

void
load_fxn(void)
{
#ifdef DEBUG_MODE
  printk("load_fxn : starting\n");
#endif

#ifdef DEBUG_MODE
  if(function_info[shmem->fid].active)
#if defined RTLINUX
```

```
    rtl_printf("ERROR : Function %d is already loaded\n",shmem->fid);
#elif defined RTAI
    rt_printk("ERROR : Function %d is already loaded\n",shmem->fid);
#endif
  else {
#endif

  function_info[shmem->fid].main_fxn = shmem->main_fxn;
  function_info[shmem->fid].update_fxn = shmem->update_fxn;
  function_info[shmem->fid].active = 1;
  function_info[shmem->fid].paused = 1;

  function_info[shmem->fid].update_fxn(sync_fifo_data,shmem->fid,1);

#ifdef DEBUG_MODE
  }
#endif

  shmem->main_fxn = NULL;
  shmem->update_fxn = NULL;
  shmem->fid = -1;
  shmem->state_flag = NO_UPDATE;

#ifdef DEBUG_MODE
  printk("load_fxn : finished\n");
#endif
}

void
unload_fxn(void)
{
#ifdef DEBUG_MODE
  printk("unload_fxn : starting\n");
#endif

#ifdef DEBUG_MODE
  if(!function_info[shmem->fid].active)
#if defined RTLINUX
    rtl_printf("ERROR : Function %d is not loaded\n",shmem->fid);
#elif defined RTAI
    rt_printk("ERROR : Function %d is not loaded\n",shmem->fid);
#endif
  else
#endif

  function_info[shmem->fid].active = 0;

  shmem->fid = -1;
  shmem->state_flag = NO_UPDATE;

#ifdef DEBUG_MODE
  printk("unload_fxn : finished\n");
```

```
#endif
}

void
pause_fxn(void)
{
#ifdef DEBUG_MODE
  printk("pause_fxn : starting\n");
#endif

#ifdef DEBUG_MODE
  if(function_info[shmem->fid].paused)
#if defined RTLINUX
    rtl_printf("ERROR : Function %d is already paused\n",shmem->fid);
#elif defined RTAI
    rt_printk("ERROR : Function %d is already paused\n",shmem->fid);
#endif
  else
#endif

  function_info[shmem->fid].paused = 1;

  shmem->fid = -1;
  shmem->state_flag = NO_UPDATE;

#ifdef DEBUG_MODE
  printk("pause_fxn : finished\n");
#endif
}

void
unpause_fxn(void)
{
#ifdef DEBUG_MODE
  printk("unpause_fxn : starting\n");
#endif

#ifdef DEBUG_MODE
  if(!function_info[shmem->fid].paused)
#if defined RTLINUX
    rtl_printf("ERROR : Function %d is not paused\n",shmem->fid);
#elif defined RTAI
    rt_printk("ERROR : Function %d is not paused\n",shmem->fid);
#else
#error "Must #define RTLINUX or RTAI"
#endif
  else
#endif

  function_info[shmem->fid].paused = 0;

  shmem->fid = -1;
```

```
        shmem->state_flag = NO_UPDATE;

       #ifdef DEBUG_MODE
        printk("unpause_fxn : finished\n");
       #endif
       }

void
zero_daq_output(void)
{
  int i;
  int ao_range_zero[NUM_AO_RANGES] = AO_INT_ZERO;
  lsampl_t doData[2];

#ifdef DEBUG_MODE
  printk("zero_daq_output : starting\n");
#endif

  do_data_insn[0] = 0x00FF;
  do_data_insn[1] = 0;
  comedi_do_insn(daq_dev, &(ddqo[0]));

  for(i=0;i<REAL_MAX_AO_CHANNELS;i++) {
    ao_real_data[i]    = ao_range_zero[0];

    daqo[i].insn       = INSN_WRITE;
    daqo[i].subdev     = ao_subd;
    daqo[i].n          = 1;
    daqo[i].data       = ao_real_data+i;
    daqo[i].chanspec   = CR_PACK(i,0,AREF_GROUND);

    comedi_do_insn(daq_dev,&daqo[i]);
   }

  for(i=0;i<MAX_AO_CHANNELS;i++) {
    ao_real_data[i%2]  = ao_range_zero[0];

    daqo[i].insn       = INSN_WRITE;
    daqo[i].subdev     = ao_subd;
    daqo[i].n          = 1;
    daqo[i].data       = ao_real_data+(i%2);
    daqo[i].chanspec   = CR_PACK(i,0,AREF_GROUND);
  }

  /* doing sample and hold do ... */
  for (i=0;i<MAX_AO_CHANNELS;i++) {
    doData[1] = 1 << i;
    comedi_do_insn(daq_dev,&ddqo[0]);
  }

#ifdef DEBUG_MODE
  printk("zero_daq_output : finished\n");
```

```
#endif
}

void
zero_ddq_output(void)
{
  int i;
  lsampl_t data[2];

#ifdef DEBUG_MODE
  printk("zero_ddq_output : starting\n");
#endif

  data[0] = 0x00FF;
  data[1] = 0;
  for(i=0; i<MAX_DO_CHANNELS; i++) {
    ddqo[i].insn     = INSN_BITS;
    ddqo[i].subdev   = do_subd;
    ddqo[i].n        = 2;
    ddqo[i].data     = data;
    ddqo[i].chanspec = i;

    comedi_do_insn(daq_dev, &(ddqo[i]));
  }
#ifdef DEBUG_MODE
  printk("zero_ddq_output : finished\n");
#endif
}
```

```
void
cleanup_module(void)
{
#ifdef DEBUG_MODE
  printk("cleanup_module : starting\n");
#endif

  cleanup_rt_thread();
  cleanup_comedi();
  cleanup_shmem();

#ifdef DEBUG_MODE
  printk("cleanup_module : finished\n");
#endif
}

void
cleanup_rt_thread(void)
{
#ifdef DEBUG_MODE
  printk("cleanup_rt_thread : starting\n");
#endif
```

```
#if defined RTLINUX
  pthread_delete_np(rt_thread);
#elif defined RTAI
  rt_task_delete(&rt_thread);
#else
#error "Must #define RTLINUX or RTAI"
#endif

#ifdef DEBUG_MODE
  printk("cleanup_rt_thread : finished\n");
#endif
}

void
cleanup_comedi(void)
{
#ifdef DEBUG_MODE
  printk("cleanup_comedi : starting\n");
#endif

  comedi_cancel(daq_dev,ai_subd);
  comedi_unlock(daq_dev,ao_subd);
  comedi_unlock(daq_dev,ai_subd);

comedi_unlock(daq_dev,do_subd);

  comedi_close(daq_dev);

#ifdef DEBUG_MODE
  printk("cleanup_comedi : finished\n");
#endif
}
void
cleanup_shmem(void)
{
#ifdef DEBUG_MODE
  printk("cleanup_shmem : starting\n");
#endif

#if defined RTLINUX
  mbuff_free(SHM_ADDR,(void *)shmem);
#elif defined RTAI
  //  rtai_kfree((void *)shmem);
  rtai_kfree(nam2num(SHM_ADDR));
#else
#error "Must #define RTLINUX or RTAI"
#endif
#ifdef DEBUG_MODE
  printk("cleanup_shmem : finished\n");
#endif
}
```

APÊNDICE 3 - GNU GENERAL PUBLIC LICENSE - Version 2, June 1991

GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you". Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program. You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notice stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that

choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS